TUGAS AKHIR – KI141502

PERANCANGAN DAN ANALISIS ALGORITMA GRAPH DINAMIS PADA PENYELESAIAN PERMASALAHAN SISTEM PERSAMAAN KONGRUEN YANG DINAMIS

Andy William
5111100095

Dosen Pembimbing 1
Ahmad Saikhu, S.Si., M.T.

Dosen Pembimbing 2
Rully Soelaiman, S.Kom., M.Kom.

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya 2015

UNDERGRADUATE THESIS - KI141502

ALGORITHM DESIGN AND ANALYSIS FOR DYNAMIC GRAPH ON DYNAMIC CONGRUENCE EQUATION SYSTEM PROBLEM

Andy William
5111100095

Supervisor 1
Ahmad Saikhu, S.Si., M.T.

Supervisor 2
Rully Soelaiman, S.Kom., M.Kom.

DEPARTMENT OF INFORMATICS
Faculty of Information Technology
Institut Teknologi Sepuluh Nopember
Surabaya 2015

LEMBAR PENGESAHAN

PERANCANGAN DAN ANALISIS ALGORITMA GRAPH DINAMIS PADA PENYELESAIAN PERMASALAHAN SISTEM PERSAMAAN KONGRUEN YANG DINAMIS

TUGAS AKHIR

Diajukan Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Dasar dan Terapan Komputasi
Program Studi S-1 Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember

Oleh:
Andy William
NRP 5111100095

Disetujui oleh Pembimbing Tugas Akhir:

1. Ahmad Saikhu, S.Si., M.T. .....
   (NIP. *197512202001121002*) (Pembimbing 1)

2. Rully Soelaiman, S.Kom., M.Kom. .....
   (NIP. *197002131994021001*) (Pembimbing 2)

SURABAYA
Juni, 2015

ALGORITHM DESIGN AND ANALYSIS FOR DYNAMIC GRAPH ON DYNAMIC CONGRUENCE EQUATION SYSTEM PROBLEM

ABSTRACT

Name : **Andy William**
NRP : **5111100095**
Department : **Department of Informatics – FTIf ITS**
Supervisor I : **Ahmad Saikhu, S.Si., M.T.**
Supervisor II : **Rully Soelaiman, S.Kom., M.Kom.**

Congruence equation is the base of many mathematics theorems. Multiple congruence equation that refer to each other is called congruence equation system. Solution for a normal congruence equation system is not enough, because in practice there will be a lot of changes on the observed system.

This constantly changing system is called dynamic congruence equation system. In this problem, the answer of each equation can be calculated without calculating from scratch if some modification occurs, including addition or variable changes on the equations.

Since every congruence equation refers to other congruence equation, the congruence equation system can be remodeled into graph. A change in the equation can be represented as edge deletion and edge insertion in the graph

In this undergraduate thesis will be implemented the solution for dynamic congruence equation system problem using dynamic graph, specifically link cut tree. The solution is able to find the answer or change every congruence equation in dynamic congruence equation system. Furthermore, the solution has a linear space complexity and O(log n) time complexity for each operation.

Keyword: Dynamic Congruence Equation System, Dynamic Graph, Edge Deletion, Edge Insertion.

PERANCANGAN DAN ANALISIS ALGORITMA GRAPH DINAMIS PADA PENYELESAIAN PERMASALAHAN SISTEM PERSAMAAN KONGRUEN YANG DINAMIS

ABSTRAK

Name : **Andy William**
NRP : **5111100095**
Jurusan : **Teknik Informatika – FTIf ITS**
Dosen Pembimbing I : **Ahmad Saikhu, S.Si., M.T.**
Dosen Pembimbing II : **Rully Soelaiman, S.Kom., M.Kom.**

Persamaan kongruen merupakan dasar dari banyak teorema pada ilmu matematika. Kumpulan persamaan kongruen yang saling merujuk satu sama lain disebut sistem persamaan kongruen. Solusi umum untuk permasalahan sistem persamaan kongruen tidak cukup, karena pada kenyataannya seringkali terjadi perubahan pada sistem persamaan yang sedang diteliti.

Sistem persamaan kongruen yang dapat berubah-ubah disebut sistem persamaan kongruen dinamis. Pada permasalahan sistem kongruen dinamis, hasil persamaan harus dapat dihitung tanpa mengulang dari awal jika terjadi modifikasi pada persamaan, termasuk penambahan atau perubahan persamaan.

Persamaan kongruen selalu mengacu persamaan lain, sehingga sistem persamaan dapat dimodelkan menjadi graph. Perubahan persamaan dapat dilambangkan sebagai penghapusan edge dan penyisipan edge pada graph. Graph ini dinamakan link cut tree.

Pada Tugas Akhir ini diimplementasikan solusi untuk permasalahan sistem persamaan kongruen dinamis dengan menggunakan graph dinamis, yaitu link cut tree. Solusi tersebut dapat menemukan hasil atau mengubah setiap persamaan kongruen pada sistem persamaan kongruen dinamis, serta mempunyai kompleksitas space linear dan kompleksitas time O(log n) untuk setiap operasi.

Kata kunci: Sistem Persamaan Kongruen Dinamis, Graph Dinamis, Penghapusan Edge, Penyisipan Edge.

KATA PENGANTAR

Puji syukur saya panjatkan kepada Tuhan Yang Maha Esa yang telah memberikan rahmat dan karunia-Nya, sehingga saya dapat menyelesaikan Tugas Akhir yang berjudul:

PERANCANGAN DAN ANALISIS ALGORITMA GRAPH DINAMIS PADA PENYELESAIAN PERMASALAHAN SISTEM PERSAMAAN KONGRUEN YANG DINAMIS

Tugas Akhir ini diajukan untuk memenuhi salah satu syarat memperoleh gelar Sarjana Komputer di Jurusan Teknik Informatika Fakultas

Dengan selesainya Tugas Akhir ini saya harap apa yang telah saya kerjakan dapat memberikan manfaat bagi perkembangan ilmu pengetahuan serta bagi saya selaku penulis Tugas Akhir ini.

Saya selaku penulis menyampaikan terima kasih kepada semua pihak yang telah memberikan dukungan baik secara langsung maupun tidak langsung selama saya menyelesaikan Tugas Akhir antara lain:

1. Tuhan Yang Maha Esa yang telah memberikan rahmat dan karunia-Nya sehingga saya dapat menyelesaikan Tugas Akhir dengan baik.
2. Ibu Dr. Eng. Nanik Suciati, S.Kom., M.Kom. selaku Ketua Jurusan Teknik Informatika Fakultas Teknologi Informasi Institut Teknologi Sepuluh Nopember.
3. Bapak Radityo Anggoro, S.Kom., M.Sc. selaku Koordinator Tugas Akhir.
4. Bapak Ahmad Saikhu, S.Si., M.T. selaku Dosen Pembimbing I yang telah memberikan dukungan dan bimbingan kepada saya dalam menyelesaikan Tugas Akhir.
5. Bapak Rully Soelaiman, S.Kom., M.Kom. selaku Dosen Pembimbing II yang telah memberikan ilmu, motivasi, kritik, dan saran kepada saya dalam menyelesaikan Tugas

Akhir.

6. Seluruh teman saya dalam menempuh perkuliahan di Jurusan Teknik Informatika Fakultas Teknologi Informasi Institut Teknologi Sepuluh Nopember.
7. Keluarga saya yang telah memberikan doa dan dukungan baik secara moral maupun finansial selama saya menyelesaikan Tugas Akhir.
8. Seluruh pihak yang tidak bisa saya sebutkan satu persatu yang telah memberikan dukungan selama saya menyelesaikan Tugas Akhir.

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR KODE SUMBER

BAB I
PENDAHULUAN

Pada bab ini penulis menjelaskan tentang latar belakang, rumusan masalah, batasan masalah, tujuan, metodologi, dan sistematika penulisan Tugas Akhir

1.1 Latar Belakang

Latar belakang Tugas Akhir ini adalah permasalahan *Dynamic Congruence Equation System* pada situs SPOJ. Permasalahan tersebut adalah mencari hasil dari sistem persamaan kongruen. Tetapi, variabel dari setiap persamaan dalam sistem dapat berubah sewaktu-waktu.

Hal tersebut memunculkan permasalahan sistem persamaan kongruen yang dinamis. Permasalahan yang terjadi adalah bagaimana solusi yang efisien untuk mempertahankan kondisi sistem jika terjadi modifikasi pada persamaan, termasuk penambahan atau perubahan persamaan. Solusi yang paling mudah adalah melakukan komputasi ulang dari nol setiap terjadi modifikasi pada sistem, tetapi hal tersebut tentu saja tidak efisien.

Pada Tugas Akhir ini diimplementasikan solusi untuk permasalahan sistem persamaan kongruen yang dinamis dengan menggunakan *graph* dinamis. Solusi tersebut mempunyai kompleksitas waktu optimal untuk setiap kueri pengambilan hasil dan perubahan persamaan.

1.2 Rumusan Masalah

Rumusan masalah yang diangkat dalam Tugas Akhir ini adalah:

1. Bagaimana desain algoritma dan struktur data yang efisien untuk permasalahan sistem persamaan kongruen yang dinamis?
2. Bagaimana implementasi algoritma dan struktur data

yang efisien berdasarkan desain yang telah dilakukan?
3. Bagaimana uji coba untuk mengetahui kebenaran dan kinerja dari implementasi yang telah dilakukan?

1.3 Batasan Masalah

Permasalahan yang dibahas pada Tugas Akhir ini memiliki beberapa batasan, yaitu sebagai berikut:

1. Implementasi dilakukan dengan bahasa pemrograman C++.
2. Banyaknya persamaan kongruen tidak lebih dari 30000.
3. Banyaknya operasi mengubah dan menghitung persamaan tidak lebih dari 100000.
4. Persamaan bisa tidak memiliki solusi atau memiliki banyak solusi.

1.4 Tujuan

Tujuan dari Tugas Akhir ini adalah mengimplementasikan solusi untuk permasalahan sistem persamaan kongruen dinamis dan melakukan uji coba untuk mengetahui kebenaran dan kinerja dari implementasi yang telah dilakukan.

1.5 Metodologi

Berikut metodologi yang digunakan dalam Tugas Akhir ini:

1. Penyusunan proposal Tugas Akhir
   Pada tahap ini dilakukan penyusunan proposal tugas akhir yang berisi perancangan dan analisis algoritma *graph* dinamis untuk permasalahan sistem persamaan kongruen dinamis.
2. Studi literatur
   Pada tahap ini dilakukan pencarian informasi dan studi literatur mengenai *graph* dinamis yang diperlukan untuk penyelesaian permasalahan sistem persamaan kongruen dinamis. Materi-materi tersebut didapatkan dari internet maupun buku acuan.

3. Desain
   Pada tahap ini dilakukan desain algoritma serta struktur data dari solusi untuk permasalahan sistem persamaan kongruen dinamis.
4. Implementasi
   Pada tahap ini dilakukan implementasi solusi untuk permasalahan sistem persamaan kongruen dinamis.
5. Uji coba dan evaluasi
   Pada tahap ini dilakukan uji coba untuk menguji kebenaran dan kinerja dari implementasi yang telah dilakukan. Pengujian kebenaran dilakukan dengan beberapa masukan kasus uji, kemudian keluaran program dibandingkan dengan hasil pengujian dari implementasi yang lain. Pengujian kinerja dilakukan dengan membandingkan kompleksitas yang didapat dari hasil pengujian dengan kompleksitas yang didapat dari hasil analisis. Selain itu dilakukan evaluasi untuk mencari bagian-bagian yang masih bisa dioptimasi dan melakukan perbaikan jika terdapat kesalahan.
6. Penyusunan buku Tugas Akhir
   Pada tahap ini dilakukan penyusunan buku Tugas Akhir yang berisi dokumentasi mengenai solusi untuk permasalahan sistem persamaan kongruen dinamis.

1.6 Sistematika Penulisan

Berikut sistematika penulisan buku Tugas Akhir ini

1. BAB I: PENDAHULUAN
   Bab ini berisi latar belakang, rumusan masalah, batasan masalah, tujuan, metodologi, dan sistematika penulisan Tugas Akhir.
2. BAB II: TINJAUAN PUSTAKA
   Bab ini berisi dasar teori mengenai permasalahan dan algoritma yang digunakan dalam Tugas Akhir.
3. BAB III: DESAIN
   Bab ini berisi desain algoritma serta struktur data yang

digunakan dalam Tugas Akhir.

4. BAB IV: IMPLEMENTASI
   Bab ini berisi implementasi berdasarkan desain algoritma serta struktur data yang telah dilakukan.
5. BAB V: UJI COBA DAN EVALUASI
   Bab ini berisi uji coba dan evaluasi dari implementasi yang telah dilakukan.
6. BAB VI: KESIMPULAN DAN SARAN
   Bab ini berisi kesimpulan dari hasil uji coba yang telah dilakukan dan saran mengenai hal-hal yang masih bisa dikembangkan.

BAB II
TINJAUAN PUSTAKA

Pada bab ini penulis menjelaskan tentang beberapa tinjauan pustaka mengenai permasalahan dan algoritma yang digunakan dalam Tugas Akhir

2.1 Definisi Graph

Sebuah *graph G=(V,E)* terdiri dari himpunan *vertex* V dan himpunan *edge* E. Sebuah *edge(u,v)* terdiri dari pasangan *vertex* u dan *vertex* v. *vertex* u dan *vertex* v disebut *endpoint* dari sebuah *edge(u,v)*. Jika terdapat *edge(u,v)* dalam sebuah *graph*, maka *vertex* u bisa disebut *neighbor* dari *vertex* v dan *vertex* v disebut *neighbor* dari *vertex* u.

Graph berarah merupakan *graph* dimana *edge(u,v)* dan *edge(v,u)* dianggap berbeda, *edge(u,v)* artinya terdapat *edge* ke *vertex* v dari *vertex* sumber u sedangkan *edge(v,u)* artinya terdapat *edge* ke *vertex* tujuan u dari *vertex* sumber v. *Graph* tidak berarah merupakan *graph* dimana *edge(u,v)* dan *edge(v,u)* dianggap sama.

Sebuah *edge(u,v)* dikatakan *inedge* dari sebuah *vertex* v dan *outedge* dari sebuah *vertex* u. Banyaknya *inedge* pada *vertex* v disebut *indegree* dan banyaknya *outedge* pada *vertex* v disebut *outdegree*.

2.2 Definisi Tree

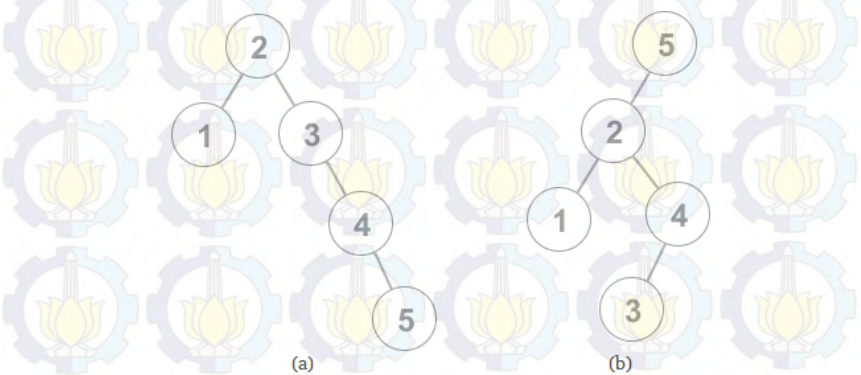
Tree adalah sebuah *graph* tidak berarah *G* setiap pasang *vertex* pada *G* terhubung dengan tepat satu jalan. Sebuah *tree* dapat memiliki sebuah *root*, yaitu *vertex* yang dilambangkan sebagai *vertex* awal. Tree seperti ini disebut *rooted tree*. Sebuah *graph* berarah *DG* dapat diperlakukan seperti tree, apabila setiap *vertex* pada *DG* hanya memiliki satu jalan untuk mencapai *root(DG)*.

Pada *rooted tree T*, *vertex root* dilambangkan sebagai *root(T)*. Setiap *vertex* lain posisinya dihitung relatif dari *root(T)*. Untuk setiap *edge(u,v)*, *vertex* yang lebih dekat dengan *root(T)* dinamakan *parent vertex*, sedangkan *vertex* yang lebih jauh dari *root(T)* dinamakan *child vertex* pada hubungan tersebut. Apabila *parent(v)* adalah u, maka pada himpunan *child(u)* terdapat v.

2.3 Splay Tree dan Lazy Splay

Sebuah *rooted tree T* dimana setiap *vertex* v pada *T* memiliki maksimal dua *child vertex* yaitu *lchild(v)* dan *rchild(v)* serta *lchild(v)* nilainya selalu lebih kecil dari *rchild(v)* disebut *binary search tree*. Umumnya, *lchild(v)* diilustrasikan berada di sebelah kiri dari *rchild(v)*. *Splay tree* adalah sebuah *binary search tree* yang strukturnya cenderung seimbang, berarti *vertex* terjauh dari *root* diusahakan sedekat mungkin, sehingga setiap *vertex* dapat diakses dengan waktu yang relatif sama., yaitu *O(log N)*.

Apabila *vertex v* diakses pada *splay tree S*, maka dilakukan operasi *splay(v)*. Pada s*play(v),* struktur S diubah sehingga *root(S)=v* dengan melakukan rotasi pada *vertex*. Teknik ini dinamakan *splaying*. Ilustrasi *splay tree* ditunjukkan pada Gambar 2.1.

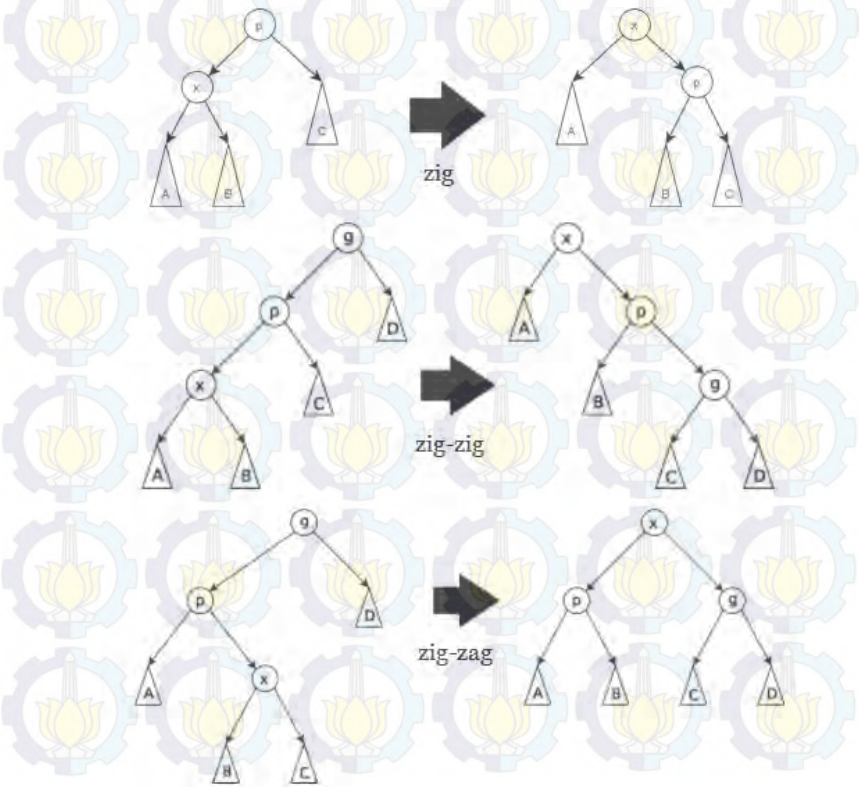


Gambar 2.1 Ilustrasi splay tree (a) dan setelah vertex 5 diakses (b)

Pada S, seluruh *vertex* yang berada dalam *subtree s* dapat diubah nilainya tanpa mengubah satu persatu *vertex*. Perubahan yang ingin dilakukan hanya disimpan pada *root(s)*. Apabila elemen didalam s ingin diakses barulah perubahan tersebut diturunkan ke semua *child(root(s))*. Teknik ini dinamakan *lazy splaying*.

2.3.1 Algoritma Splay

Untuk menjaga keseimbangan *tree*, pada *splay tree S* perlu dilakukan operasi *splay(x)* pada *vertex* x yang diakses. Ilustrasi operasi splay yang mungkin ditunjukkan pada Gambar 2.2.



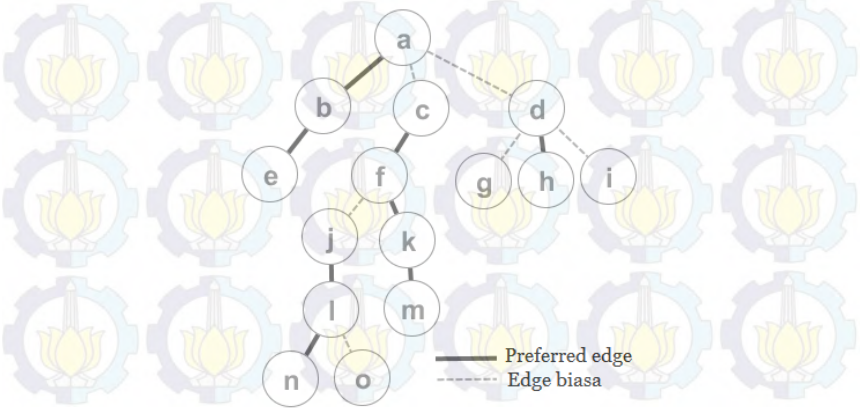
Gambar 2.2 Ilustrasi kemungkinan dari setiap iterasi splay

Setiap iterasi pada operasi *splay(x)* tergantung pada x, *parent(x)* yaitu p, serta *parent(p)* yaitu g. Apabila *root(S)=p*, maka *tree* dirotasi pada *edge(p,x)*. Operasi ini dinamakan *zig*. Apabila *root(S)≠p*, serta x dan p keduanya adalah *lchild* atau *rchild* dari *parent*nya masing-masing, maka *edge(g,p)* dirotasi kemudian *edge(p,x)* dirotasi. Operasi ini dinamakan *zig-zig*. Apabila kedua syarat diatas tidak terpenuhi, maka *edge(p,x)* dirotasi kemudian *edge(g,x)* dirotasi. Operasi ini dinamakan *zig-zag*. Operasi *splay(v)* melakukan hal ini secara berulang sampai *root(S)=x*.

2.4 Link Cut Tree

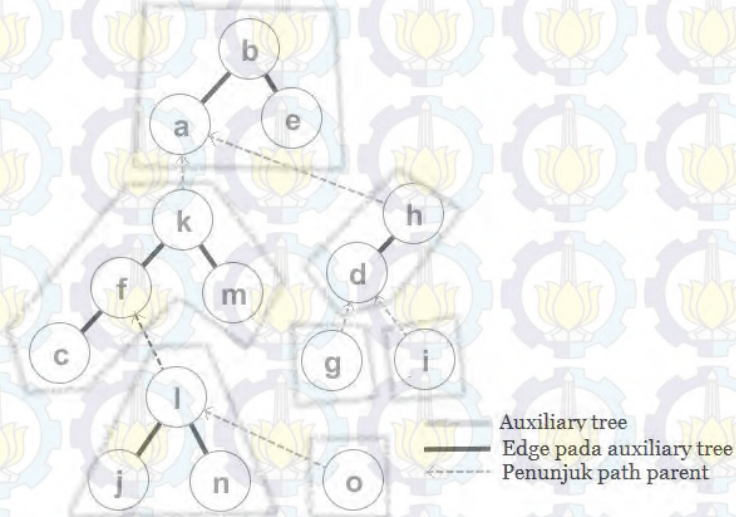
Link cut tree adalah struktur data untuk merepresentasikan kumpulan *tree*. Setiap *tree* dipisah menjadi jalur atau *path* yang linier. Kumpulan jalur tersebut dinamakan *represented tree*.

Ketika sebuah *vertex* v diakses pada *represented tree RT,* maka jalur dari v menuju *root(RT)* yaitu *path(v)* menjadi *preferred path*. Untuk setiap *vertex* v pada *RT*, *child(v)* yang berada pada jalur yang terakhir diakses dinamakan *preferred child*. *Edge* yang terhubung ke *preferred child* disebut *preferred edge*. Ilustrasi represented tree ditunjukkan pada Gambar 2.3.



Gambar 2.3 Ilustrasi represented tree

Setiap *path p* pada *RT* direpresentasikan oleh *auxiliary tree AT(v)* dalam bentuk *splay tree*, dimana *path(v)=p* dan setiap *vertex* v hanya terdapat dalam satu *path*. Untuk setiap *AT(v)*, *vertex* yang lebih dekat dengan *root(RT)* memiliki nilai yang lebih kecil, sehingga *AT* terurut berdasarkan kedalaman atau *depth* pada *RT*. Ilustrasi *auxiliary tree* ditunjukkan pada Gambar 2.4.



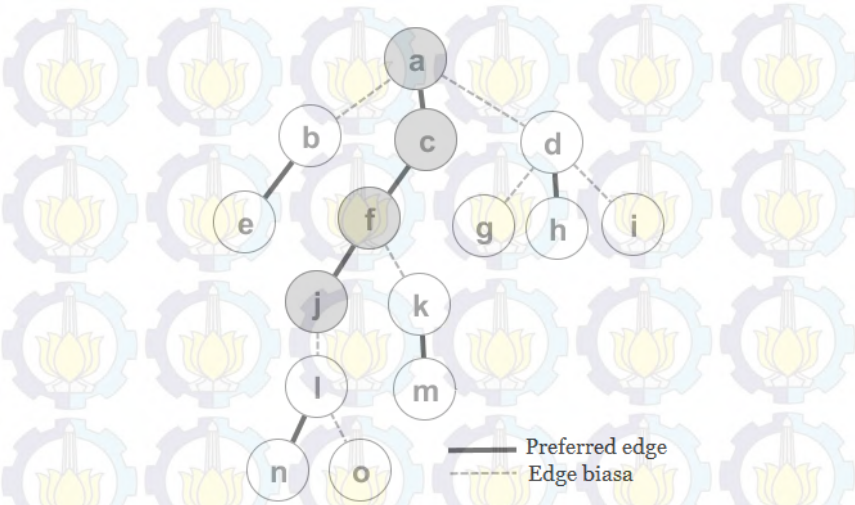
Gambar 2.4 Ilustrasi kumpulan auxiliary tree dalam bentuk splay tree

Edge(u,v) yang terhubung pada *RT* namun tidak berada pada *path* yang sama, atau dengan kata lain *path(u)≠path(v)*, seperti a dan c pada Gambar 2.3, dimana *parent(v)=u* akan memiliki penunjuk *path parent PP(AT(v))=u*.

2.4.1 Algoritma Akses

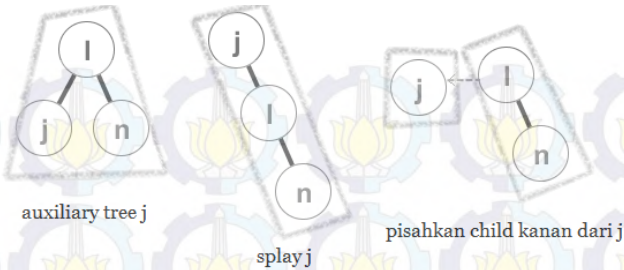
Operasi *access(v)* membuat *path* dari *vertex* v menuju *root(RT)*. Setelah *access(v)* dilakukan, *vertex* v menjadi ujung dari *path(v)*, sehingga v tidak memiliki *preferred child*. Jalur dari v menuju *root(RT)* akan sepenuhnya terdiri dari *preferred edge*. Ilustrasi *represented tree* setelah melakukan *access(j)* ditunjukkan pada

Gambar 2.5.



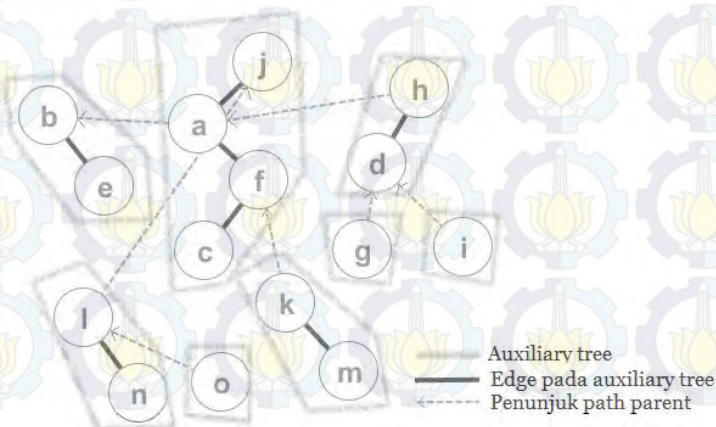
Gambar 2.5 Ilustrasi represented tree setelah melakukan access(j)

Karena v menjadi ujung dari jalur, maka v harus menjadi *vertex* paling dalam pada *auxiliary tree AT*. Setiap *vertex* yang memiliki nilai lebih dari v harus diputus dari *AT,* sehingga v menjadi *vertex* dengan nilai *depth* terbesar pada *AT*. Dalam splay tree, operasi ini dilakukan dengan cara melakukan *splay(v)*, sehingga *root(AT)=v*. Setelah itu *rchild(v)=r* dipisah menjadi *auxiliary tree AT(r)*. Karena hubungan pada *RT* tetap ada, maka *PP(AT(r))=v*. Ilustrasi pemisahan *auxiliary tree* milik j ditunjukkan pada Gambar 2.6.



Gambar 2.6 Ilustrasi pemisahan auxiliary tree milik j

Operasi dilanjutkan dengan menggabungkan *AT(v)* dengan *AT(pp)*, dimana *pp=PP(AT(v))*. Untuk menggabungkan kedua *auxiliary tree*, maka pada *AT(pp)* juga harus dilakukan operasi seperti pada paragraf sebelumnya. Selanjutnya, lakukan *rchild(pp)=v* sehingga kedua *auxiliary tree* bergabung menjadi satu, dan lakukan *splay(v)* sehingga v menjadi *root* dari *AT(v)*. *Path parent* milik v sekarang berubah menjadi *path parent* milik pp. Operasi ini dilakukan terus sampai *AT(v)* tidak memiliki *path parent*, yang berarti jalur dari v menuju *root* sudah terletak pada satu *auxiliary tree*. Ilustrasi *auxiliary tree* setelah melakukan *access(j)* ditunjukkan pada Gambar 2.7.



Gambar 2.7 Ilustrasi auxiliary tree setelah melakukan access(j)

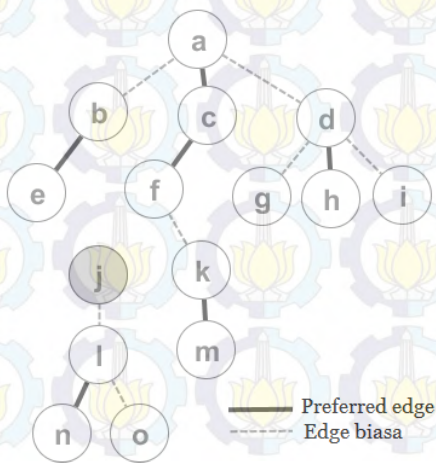
Operasi akses adalah dasar dari semua operasi lain pada link cut tree. Setiap operasi lain selalu melakukan minimal satu buah operasi akses. Operasi akses memiliki *amortized time complexity O(log n)*.

2.4.2 Algoritma Mencari Root

Operasi *findroot(v)* mencari *root* pada *RT(v)*. Lakukan *access(v)*, kemudian kembalikan *vertex* r dengan nilai terkecil pada *AT(v)*. Setelah itu lakukan operasi *splay(r)*. Pencarian nilai terkecil dapat dilakukan dengan menelusuri *left child* dari *root* secara rekursif sampai mencapai *vertex* yang tidak mempunyai *left child* lagi.

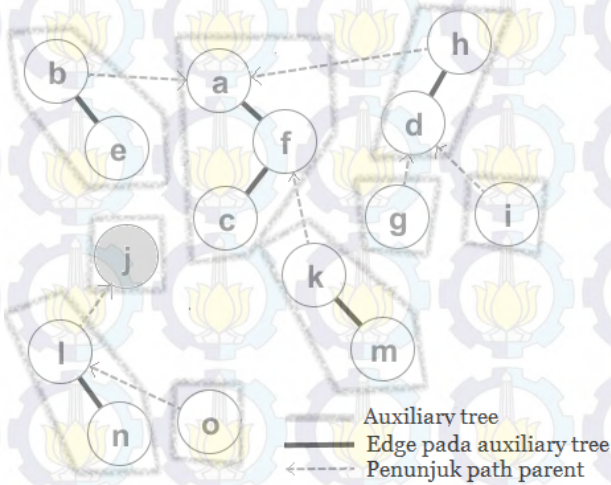
2.4.3 Algoritma Penghapusan Edge

Operasi *cut(v)* menghapus *edge(parent(v),v)* pada *RT(v)*. Setelah operasi penghapusan *edge* dilakukan, *tree* akan terpisah menjadi dua, dengan v menjadi *root* dari *tree* yang baru. Ilustrasi *represented tree* setelah melakukan *cut(j)* ditunjukkan pada Gambar 2.8.



Gambar 2.8 Ilustrasi represented tree setelah melakukan cut(j)

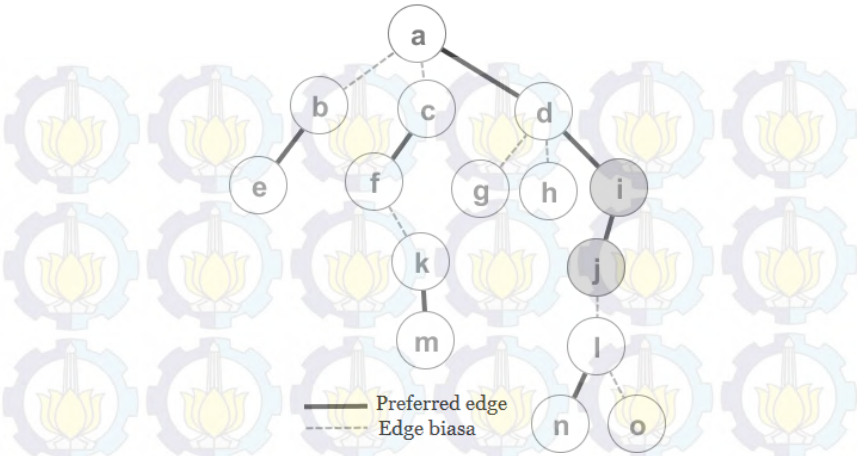
Operasi *cut(v)* dilakukan dengan melakukan operasi *access(v)*. Kemudian hapus hubungan v dengan semua *node* yang memiliki *depth* yang lebih kecil, dengan cara memindahkan *PP(v)* ke *PP(lchild(v))* dan menghapus *edge(v,lchild(v))* pada *AT(v)*. Ilustrasi *auxiliary tree* setelah melakukan *cut(j)* ditunjukkan pada Gambar 2.9.



Gambar 2.9 Ilustrasi auxiliary tree setelah melakukan cut(j)

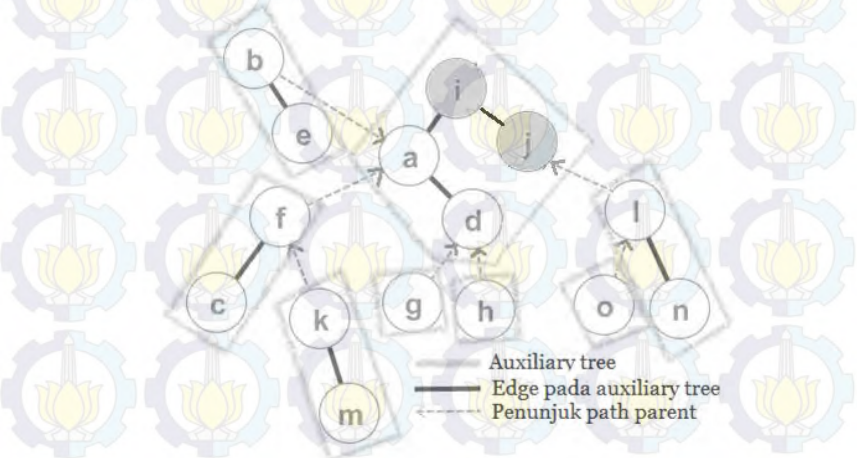
2.4.4 Algoritma Penyisipan Edge

Operasi *link(u,v)* menghubungkan *vertex* u dan v dari dua *represented tree* yang berbeda menjadi satu, dengan syarat properti *tree* harus dipertahankan. Ilustrasi *represented tree* setelah melakukan *link(j,i)* ditunjukkan pada Gambar 2.10.



Gambar 2.10 Ilustrasi represented tree setelah melakukan link(j,i)

Operasi *link(u,v)* dilakukan dengan cara melakukan *access(u)* dan *access(v)*, kemudian gabungkan *AT(v)* dengan *AT(u)* dengan cara membuat r*child(u)=v*. Ilustrasi *auxiliary tree* setelah melakukan *cut(j)* ditunjukkan pada Gambar 2.11.



Gambar 2.11 Ilustrasi auxiliary tree setelah melakukan link(j,i)

2.5 Metode Invers Modular

Sifat distributif dari modulus hanya berlaku pada penjumlahan dan perkalian. Pada pembagian, sifat tersebut tidak berlaku. Inverse modular dari sebuah bilangan a modulus m adalah x dimana $ax \equiv 1 (\text{mod } m)$. Dapat dikatakan bahwa $a^{-1} \equiv x (\text{mod } m)$.

Sebagai contoh, kita ingin mencari hasil dari $x \equiv 3^{-1} (\text{mod } 11)$. Hal ini sama dengan mencari hasil dari $3x \equiv 1 (\text{mod } 11)$. Satu-satunya hasil yang juga memenuhi $1 \leq x \leq 11$ adalah 4. Berarti, pembagian dengan 3 pada sebuah persamaan dengan modulus 11 dapat disubstitusikan menjadi perkalian dengan 4. Contoh kasusnya adalah $12/3 \equiv 12 \times 4 (\text{mod } 11)$, kedua sisi sama-sama menghasilkan hasil 4 setelah dimodulus 11.

Kegunaan dari melakukan invers modular daripada melakukan pembagian adalah perhitungan dapat dilakukan walaupun angka dapat menjadi sangat besar. Karena sifat modulus yang distributif pada perkalian, angka besar yang terdiri dari banyak perkalian dapat dihitung secara iteratif per suku. Karena pembagian dapat diubah menjadi perkalian dengan invers modular, maka angka besar yang mengandung pembagian juga dapat dihitung.

Apabila modulus pada suatu sistem telah diketahui yaitu m, maka semua invers modular dari 1 sampai m-1 dapat dihitung terlebih dahulu agar untuk perhitungan selanjutnya tidak perlu menghitung ulang. Karena pembagian dengan a sama dengan perkalian dengan invers a, maka $a^{-n} \equiv x^n (\text{mod } m)$. Metode paling cepat untuk melakukan perhitungan semua invers modular dari 1 sampai m-1 adalah mencari nilai n sehingga untuk n=1 sampai m-1 nilai a^n berbeda semua. Untuk modulus 10007, nilai n yang cocok adalah 5, dengan nilai x 4003.

2.6 Sphere Online Judge (SPOJ)

Sphere Online Judge (SPOJ) adalah sistem penilaian *online* yang berisi permasalahan pemrograman yang dapat diselesaikan oleh

pengguna dengan mengirim kode sumber program yang berisi solusi dari permasalahan. Setiap permasalahan mempunyai format masukan yang diberikan dan format keluaran yang diminta. Selain itu setiap permasalahan juga mempunyai batasan tertentu termasuk lingkungan penilaian, batasan waktu, batasan memori, dan batasan dari permasalahan yang dideskripsikan.

Kode sumber program yang diterima sistem dikompilasi dan dijalankan pada lingkungan penilaian sistem. Program diuji menggunakan masukan dari berkas masukan permasalahan kemudian hasil keluaran program dibandingkan dengan berkas keluaran permasalahan. Sistem akan memberikan umpan balik kepada pengguna antara lain:

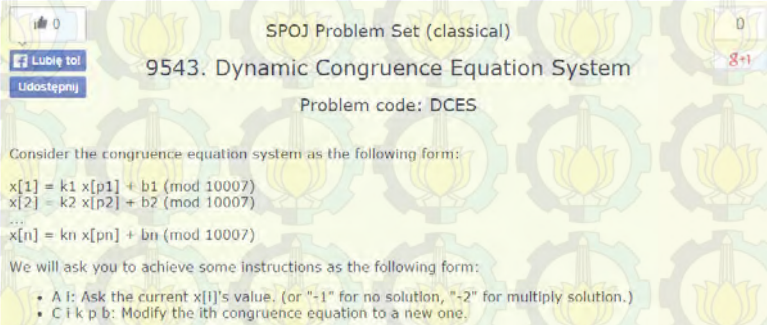
1. ***Accepted***, artinya program tidak melanggar batasan yang diberikan dan hasil keluaran program sama dengan berkas keluaran permasalahan.
2. ***Wrong Answer***, artinya hasil keluaran program tidak sama dengan berkas keluaran permasalahan.
3. ***Time Limit Exceeded***, artinya program melanggar batasan waktu yang diberikan.
4. ***Memory Limit Exceeded***, artinya program melanggar batasan memori yang diberikan.
5. ***Runtime Error***, artinya terjadi *error* pada saat program dijalankan.
6. ***Compile Error***, artinya terjadi *error* pada saat kompilasi kode sumber program.

Selain itu sistem juga memberikan umpan balik kepada pengguna mengenai waktu dan memori yang dibutuhkan program pada saat diuji menggunakan masukan dari berkas masukan permasalahan.

2.7 Permasalahan Dynamic Congruence Equation System pada SPOJ

Salah satu permasalahan yang terdapat pada SPOJ adalah *Dynamic Congruence Equation System* yang mempunyai nomor

9543 dan kode DCES. Deskripsi permasalahan *Dynamic Congruence Equation* ditunjukkan pada Gambar 2.12.



Gambar 2.12 Deskripsi permasalahan Dynamic Congruence Equation System

Pada permasalahan tersebut, diberikan N buah persamaan kongruen dalam bentuk $x[i] = k_i \times x[p_i] + b_i (\mathrm{mod}\, 10007)$. Kemudian, ada Q buah operasi yang harus dilakukan pada sistem persamaan tersebut. Operasi yang mungkin dilakukan adalah:

1. Diberikan i, keluarkan hasil pada persamaan x[i].
2. Diberikan i, k, p, dan b, ubah nilai persamaan x[i] sehingga $x[i] = k \times x[p] + b (\mathrm{mod}\, 10007)$.

Berikut adalah format masukan dari permasalahan:

1. Baris pertama berisi sebuah bilangan bulat N.
2. N baris berikutnya masing-masing berisi 3 buah bilangan bulat k_i, p_i, dan b_i berurutan dari i=1 sampai N.
3. Baris berikutnya berisi sebuah bilangan bulat Q.
4. Q baris berikutnya masing-masing berisi deskripsi operasi yang dilakukan antara lain:
   a) 'A' i artinya keluarkan hasil pada persamaan x[i].
   b) 'C' i k p b artinya ubah nilai persamaan x[i] sehingga $x[i] = k \times x[p] + b (\mathrm{mod}\, 10007)$.

Berikut merupakan format keluaran dari permasalahan tersebut:

1. Untuk setiap operasi yang berawalan 'A', keluaran

didefinisikan pada poin 2.

2. Keluarkan hasil dari x[i]. Apabila tidak ada nilai yang memenuhi x[i], keluarkan -1. Apabila ada lebih dari satu nilai yang memenuhi x[i], keluarkan -2.

Berikut merupakan batasan dari permasalahan tersebut:

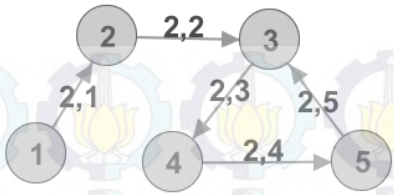
1. $1 \leq N \leq 30000$.
2. $1 \leq Q \leq 100000$.
3. $1 \leq i,p \leq 30000$.
4. $1 \leq k,b \leq 10007$.
5. Lingkungan penilaian Intel Pentium G860 3GHz.
6. Batasan waktu 0.140 detik.
7. Batasan memori 1536 MB.

Contoh masukan dan keluaran dapat dilihat pada Gambar 2.13.

Input 1:	Output 1:	Input 2:	Output 2:
5	4276	4	0
2 2 1	7141	0 1 0	-2
2 3 2	4256	1 3 0	-2
2 4 3	2126	1 4 0	-2
2 5 4		1 2 0	-1
2 3 5		6	
5		A 1	
A 1		A 2	
A 2		A 3	
C 5 3 1 1		A 4	
A 4		C 1 1 5 1	
A 5		A 1	

Gambar 2.13 Contoh masukan dan keluaran permasalahan Dynamic Congruence Equation System

Untuk menyelesaikan contoh masukan 1, persamaan dimodelkan ke dalam bentuk *graph*. Setiap persamaan dilambangkan dengan sebuah *vertex* dan persamaan yang diacu menjadi tujuan dari *edge* pada *vertex* tersebut. Representasi *graph* dari Contoh contoh masukan 1 ditunjukkan pada Gambar 2.14.



Gambar 2.14 Representasi Graph pada Contoh Masukan 1

Operasi pertama yang dilakukan adalah mencari solusi dari persamaan 1. Untuk menyelesaikannya, dilakukan substitusi. Diketahui $x[1] = 2 \times x[2] + 1$ dan $x[2] = 2 \times x[3] + 2$ maka didapatkan $x[1] = 4 \times x[3] + 5$. Apabila substitusi diteruskan, akan didapatkan hasil-hasil selanjutnya yaitu:

- $x[1] = 8 \times x[4] + 17$
- $x[1] = 16 \times x[5] + 49$
- $x[1] = 32 \times x[3] + 129$

Hasil substitusi terakhir mengandung variabel x[3] di dalamnya, yang berarti ada *cycle* pada *graph*. Solusi ini dapat digabungkan dengan solusi sebelumnya yang juga mengandung x[3]. Apabila kedua persamaan tersebut digabungkan hasilnya menjadi $4 \times x[3] + 5 = 32 \times x[3] + 129$ yang dapat disederhanakan menjadi $x[3] = -124 / 28$. Karena sistem persamaan dimodulus 10007, maka dapat ditemukan hasilnya dengan invers modular, menjadi $x[3] = -124 \times 1787 (\mathrm{mod}\, 10007)$ yang hasilnya 8573. Setelah ditemukan hasil x[3], maka x[1] dapat ditemukan hasilnya dengan memasukkan hasil x[3] pada persamaan $x[1] = 4 \times x[3] + 5$, yaitu 4276. Operasi kedua yaitu mencari hasil x[2] juga dapat ditemukan dengan memasukkan hasil x[3] pada persamaan $x[2] = 2 \times x[3] + 2$, yaitu 7141.

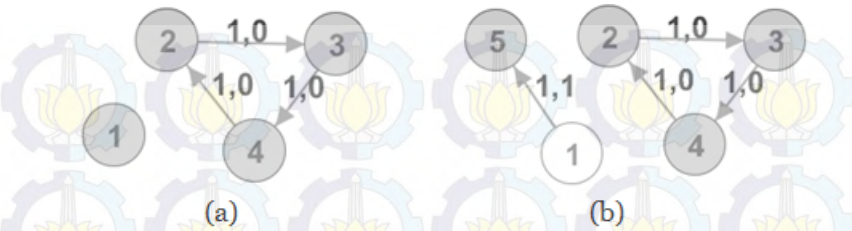
Pada operasi ketiga yaitu mengubah persamaan, *graph* yang telah terbentuk harus diubah. Representasi *graph* setelah operasi ketiga

ditunjukkan pada Gambar 2.15

Gambar 2.15 Representasi Graph setelah Operasi ke-3 Dilakukan

Hal utama yang perlu diperhatikan adalah sebagian besar *graph* tidak berubah, sehingga hampir semua persamaan yang sebelumnya telah ditemukan dapat digunakan kembali. Dari persamaan yang telah ditemukan, persamaan yang berubah hanya persamaan terakhir, yaitu $x[1] = 32 \times x[3] + 129$ yang berubah menjadi $x[1] = 48 \times x[1] + 65$. Karakteristik ini nantinya akan dieksploitasi menggunakan *link cut tree* untuk mempertahankan bagian lain dari *graph* apabila ada suatu bagian yang berubah.
Seperti sebelumnya, hasil yang dicari dahulu adalah hasil pada *cycle*, pada *graph* yang baru yaitu x[1]. Persamaan terakhir yang didapatkan yaitu $x[1] = 48 \times x[1] + 65$ sudah cukup untuk mencari solusi dari x[1], yaitu $x[1] = -65 / 47$. Dengan invers modular, persamaan tersebut dapat diubah menjadi $x[1] = -65 \times 2555 (\mathrm{mod}\, 10007)$ sehingga didapatkan hasil x[1] yaitu 4044. Setelah didapatkan hasil x[1], dua operasi setelahnya yaitu mencari hasil x[4] dan x[5] tinggal memasukkan nilai x[1] pada persamaan masing-masing, yaitu $x[4] = 6 \times x[1] + 6$ dan $x[5] = 3 \times x[1] + 1$, sehingga ditemukan hasilnya yaitu 4256 dan 2126.

Pada contoh masukan 2, didemonstrasikan beberapa hasil yang tidak valid. Representasi *graph* pada contoh masukan 2 ditunjukkan pada Gambar 2.16.



Gambar 2.16 Representasi Graph pada Contoh Masukan 2(a) dan Setelah Operasi ke-5 Dijalankan(b)

Persamaan pertama masih dapat dikerjakan walaupun merujuk ke dirinya sendiri, karena konstanta dan variabelnya bernilai 0, sehingga hasil persamaan 1 adalah 0. Persamaan 2 sampai 4 memiliki banyak solusi, karena relasinya $x[2] = x[3] = x[4]$ sehingga dapat diisi nilai berapapun asalkan nilainya sama semua. Setelah operasi ke-5 dijalankan, persamaan 1 merujuk ke persamaan yang belum terdefinisi, sehingga hasilnya tidak terdefinisi.

Untuk melakukan operasi pada sistem persamaan diperlukan informasi dari persamaan-persamaan yang terhubung baik secara langsung maupun rekursif. Diperlukan struktur data yang efisien dan mampu menyimpan atau memodifikasi properti dari semua *vertex* dalam suatu jalur. Struktur data *link cut tree* dipilih karena karakter utamanya yang melakukan operasi pada tree tidak berdasarkan *subtree*, melainkan berdasarkan jalur.

BAB III
DESAIN

Pada bab ini penulis menjelaskan tentang desain algoritma serta struktur data yang digunakan dalam Tugas Akhir.

3.1 Deskripsi Umum Sistem

Sistem menerima masukan berupa persamaan kongruen. Setelah itu, sistem memodelkan persamaan menjadi bentuk *link cut tree*. Kemudian sistem menerima masukan berupa operasi.

Terdapat dua jenis operasi yang dapat dilakukan yaitu:

1. Kueri nilai persamaan dengan memberikan index persamaan.
2. Mengubah konstanta dan variabel pada persamaan dengan memberikan index dan nilai persamaan.

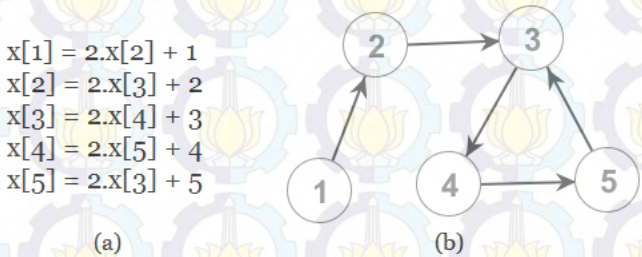
Untuk setiap kueri nilai di suatu persamaan, dikeluarkan nilai persamaan tersebut. Untuk setiap perubahan persamaan, dilakukan operasi *cut* dan *link* pada *link cut tree*. *Pseudocode* fungsi Main ditunjukkan dalam Gambar 3.1.

```
Main()
1.  Preprocess()
2.  input number of congruence equation
3.  input congruence equations
4.  convert equations to link cut tree
5.  input number of operations
6.  while number of operation > 0
7.      input type of operation
8.      if type of operation is ‘query’
9.          input query index
10.         output equation value of query index
11.     else if type of operation is ‘change’
12.         input change index, k, p, and b
13.         change value of index with k, p, and b
14.     decrement number of operation
```

Gambar 3.1 Pseudocode Fungsi Main

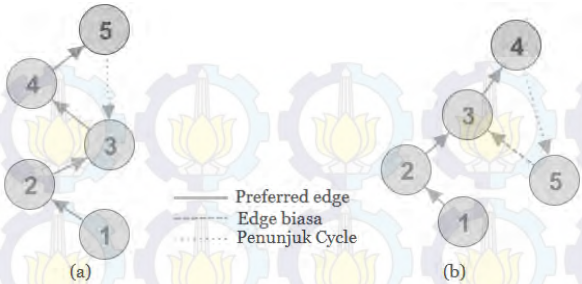
3.2 Desain Struktur Data

Setiap persamaan kongruen yang diberikan mengacu ke sebuah persamaan kongruen lain. Properti ini mirip dengan *tree*. Setiap persamaan dijadikan sebuah *vertex*. Apabila persamaan x[i] mengacu kepada persamaan x[j], maka ada *edge* terarah yang menghubungkan *vertex* i ke *vertex* j. Representasi *graph* dari sistem persamaan kongruen ditunjukkan pada Gambar 3.2.



Gambar 3.2 Sistem persamaan kongruen(a) dan representasi graphnya(b)

Setiap persamaan mengacu ke satu persamaan lain, sehingga setiap *vertex* juga mengacu ke satu *vertex* lain. Properti ini mirip seperti *rooted tree*. Perbedaannya, *root* juga harus mengacu kepada persamaan lain sehingga terbentuk path berulang atau *cycle*. Agar persamaan dapat direpresentasikan dalam *tree*, maka *cycle* dihilangkan. *Edge* yang menyebabkan *cycle* tidak dibentuk secara eksplisit, melainkan dicatat secara implisit sebagai pointer. Setiap *vertex* yang terletak pada *cycle* dapat dijadikan *root*, tetapi hanya boleh ada satu *root*. Contoh *represented tree* pada persamaan di Gambar 3.2 ditunjukkan pada Gambar 3.3.



Gambar 3.3 Represented Tree dengan Vertex 5 sebagai Root(a) dan Vertex 4 sebagai Root(b)

Untuk menghitung hasil, persamaan harus ditelusuri relasi rekurensnya sampai mengacu kepada persamaan yang sudah ditelusuri sebelumnya (*cycle* pada *graph*). Oleh karena itu, struktur data yang digunakan harus mendukung proses kalkulasi dari suatu jalur dan mampu mengubah *edge* dari setiap *vertex* secara efisien.

Implementasi naif membutuhkan kompleksitas waktu *O(n)* untuk setiap operasi. Karena jumlah *vertex* dan operasi yang besar, solusi seperti ini tidak dapat diselesaikan dalam waktu yang singkat. Dengan menggunakan *link cut tree*, setiap operasi dapat dilakukan dalam kompleksitas waktu *O(log n)*.

Jika dilakukan kueri hasil persamaan pada persamaan x[i], maka pada *vertex* i harus dibentuk jalur sampai menuju *root* menggunakan *findroot(i)* kemudian lakukan operasi perhitungan pada jalur tersebut yang akan dijelaskan pada subbab 3.3.3.

Jika dilakukan perubahan pada persamaan x[i], maka harus dilakukan operasi *cut* pada *vertex* i, kemudian melakukan *link* dari *vertex* i menuju ke persamaan baru yang dituju x[i]. Setelah melakukan pemindahan *edge*, nilai dari jalur yang ditinggalkan maupun jalur yang baru harus diperbarui. Desain untuk memperbarui nilai akan dijelaskan pada subbab 3.3.2.

3.3 Desain Algoritma

Sistem terdiri dari tiga fungsi utama yaitu *Preprocess*, *Update*, dan *Getval*. Pada subbab ini dijelaskan desain algoritma dari masing-masing fungsi tersebut.

3.3.1 Desain Fungsi Preprocess

Fungsi Preprocess menghitung hasil semua invers modular untuk modulus 10007 sehingga untuk seterusnya tidak perlu mengkalkulasi ulang. Algoritma yang digunakan seperti yang dijelaskan pada subbab 2.5. *Pseudocode* fungsi Preprocess ditunjukkan dalam Gambar 3.3.

```
Preprocess()
1.  j = 1
2.  k = 1
3.  for i=0 to 10007
4.      inv[j]=k
5.      j = j*5 mod 10007
6.      K = k*4003 mod 10007
```

Gambar 3.4 Pseudocode fungsi Preprocess

3.3.2 Desain Fungsi Update

Fungsi *Update* dilakukan untuk mempertahankan atribut dari setiap *subtree* pada *auxiliary tree AT* setelah melakukan operasi rotasi atau penghapusan atau penyisipan *edge* pada *AT*. Dengan begitu, setiap *vertex* v pada *AT* selalu menyimpan total nilai gabungan pada *subtree* dengan *root* v.

Sebuah persamaan $x[i] = k_i \times x[p_i] + b_i$ dapat digabungkan dengan persamaan yang diacu, yaitu $x[j] = k_j \times x[p_j] + b_j$ dimana $j = p_i$. Penggabungan dapat menggunakan metode substitusi, sehingga setelah kedua persamaan di atas digabung maka akan menjadi $x[i] = k_i \times (k_j \times x[p_j] + b_j) + b_i$ yang dapat diubah menjadi bentuk persamaan kongruen, yaitu $x[i] = (k_i \times k_j) \times x[p_j] + (k_i \times b_j + b_i)$. *Pseudocode* fungsi *Update* ditunjukkan dalam Gambar 3.4.

```
/***
 * n is the vertex to be updated
 */
Update(node n)
1.  if lchild(n) ≠ NULL
2.      n.pathk = n.k*lchild(n).pathk mod 10007
3.      n.pathb = (n.k*lchild(n).pathb+n.b) mod 10007
4.  else
5.      n.pathk = n.k
6.      N.pathb = n.b
7.  if rchild(n) ≠ NULL
8.      n.pathk = n.pathk*rchild(n).pathk mod 10007
9.      n.pathb = (rchild(n).pathk * n.pathb +
    rchild(n).pathb) mod 10007
```

Gambar 3.5 Pseudocode fungsi Update

Baris 1 hingga baris 3 melakukan kalkulasi nilai gabungan k(*pathk*) dan nilai gabungan b(*pathb*) dari semua *left child* dari *node n*. Baris 4 hingga baris 6 membuat nilai gabungan baru yang hanya berisi sebuah *node n* sebagai nilai awal. Baris 7 hingga baris 9 melakukan kalkulasi nilai *pathk* dan nilai *pathb* dari semua *right child* dari *node n*. Pada akhir operasi *update*, *subtree S* dengan *root* n akan menyimpan nilai *pathk* dan *pathb* pada persamaan gabungan yang berawal dari *vertex* terbesar *v* pada *S* sampai *vertex* terkecil *u* pada *S*, yaitu semua persamaan pada jalur dari *v* sampai *u* pada *represented tree*.

3.3.3 Desain Fungsi Getval

Fungsi *Getval* digunakan untuk mengambil nilai dari suatu persamaan. Fungsi ini juga digunakan untuk menentukan apakah suatu persamaan tidak memiliki solusi atau memiliki lebih dari satu solusi.

Sebuah persamaan tidak akan memiliki jawaban apabila jika ditelusuri secara rekurens pada akhirnya akan merujuk pada persamaan yang belum terdifinisi. Sebuah persamaan dapat dicari

nilai pastinya apabila jika ditelusuri secara rekurens akan merujuk kembali ke persamaan itu sendiri, sehingga bentuk gabungannya menjadi $x[i] = pathk \times x[i] + pathb$. Nilai *x[i]* kemudian dapat dicari dengan menyelesaikan persamaan tersebut sehingga didapatkan nilai $x[i] = \frac{pathb}{1-pathk}$. Perlu diperhatikan apabila *pathk=1*, maka persamaan tersebut menjadi *pathb=0*. Persamaan tidak memiliki hasil apabila *pathb≠0*. Sebaliknya, persamaan selalu benar (nilai *x[i]* bisa diisi berapapun) apabila *pathb=0*.

Apabila sebuah persamaan *x[i]* memiliki solusi, maka setiap persamaan $x[j] = pathk \times x[i] + pathb$, atau dengan kata lain persamaan yang merujuk kepada *x[i],* baik secara langsung maupun melalui relasi rekurens, juga akan memiliki hasil. Walaupun *x[i]* memiliki banyak solusi, *x[j]* bisa saja hanya memiliki satu solusi apabila nilai *pathk* pada persamaan *x[j]* adalah 0, karena 0 dikali berapapun akan menjadi 0 sehingga berapapun nilai *x[i]* tidak berpengaruh. *Pseudocode* fungsi *Getval* ditunjukkan dalam Gambar 3.5

```
/***
 * n is the vertex to be queried
 */
Getval(node n)
```

```
1.  r = findroot(n)
2.  if r is undefined
3.      return -1
4.  kAll = r.pathk
5.  bAll = r.pathb
6.  c = cycle pointer of r
7.  kCycle = c.pathk
8.  bCycle = c.pathb
9.  if kCycle=1
10.     if bCycle=0
11.         return -1
```

```
12.     else if kAll≠0
13.         return -2
14. x = bCycle*inv[1-kCycle]
15. return (kAll*x+bAll) mod 10007
```

Gambar 3.6 Pseudocode fungsi Getval

Baris 2 dan 3 adalah pengecekan apabila persamaan tidak terdefinisi. Baris 9 sampai 13 adalah kasus khusus dimana pada *x[i]* nilai *pathk=1*, seperti yang dijelaskan pada dua paragraf sebelumnya. Baris 14 adalah perhitungan *x[i]* dan baris 15 adalah perhitungan *x[j],* seperti yang dijelaskan pada dua paragraf sebelumnya.

BAB IV
IMPLEMENTASI

Pada bab ini penulis menjelaskan tentang implementasi berdasarkan desain algoritma serta struktur data yang telah dilakukan.

4.1 Lingkungan Implementasi

Lingkungan implementasi yang digunakan sebagai berikut:

1. Perangkat Keras
   Processor Intel® Core™ i7-4700MQ CPU @ 2.40GHz
   RAM 15.9 GB
   64-bit Operating System, x64-based processor
2. Perangkat Lunak
   Operating System Windows 8.1 Single Language
   Integrated Development Environment Orwell Dev-C++ 5.6.30

4.2 Implementasi Fungsi Preprocess

Fungsi *preprocess* melakukan perhitungan awal seperti dijelaskan pada subbab 3.3.1. Hasil preproses disimpan dalam variabel global inv.

```
1. const int MOD = 10007;
2. int inv[10007];
3. void pre(){
4.     int i,j,k;
5.     j=k=1;
6.     for(int i=0;i<MOD;i++){
7.         inv[j]=k;
8.         j=j*5%MOD;
9.         k=k*2502%MOD;
10.    }
11. }
```

Kode Sumber 4.1 Implementasi Fungsi Preprocess

4.3 Implementasi Struct Node

Struct Node digunakan untuk merepresentasikan sebuah *vertex v* pada *link cut tree*. Pada *Struct Node* juga terdapat implementasi algoritma yang digunakan pada *link cut tree*.

```
1.  struct node{
2.      int no;
3.      node *l,*r,*p;
4.      node *pathp;
5.      node *cycle;
6.      int k,b;
7.      int pathk,pathb;
8.      node(){
9.          l=r=p=pathp=cycle=NULL;
10.         k=b=pathk=pathb=0;
11.     }
```

Kode Sumber 4.2 Implementasi Atribut dan Konstruktor Struct Node

Nilai *no* adalah index dari persamaan kongruen. *Pointer node *l* adalah anak kiri, **r* adalah anak kanan, dan **p* adalah *parent*. *Pointer node *pathp* adalah *path parent*. *Pointer node *cycle* adalah persamaan yang berulang atau *cycle*. Nilai *k* dan *b* sesuai dengan nilai *k* dan *b* persamaan dengan index *no*. Nilai *pathk* dan *pathb* adalah nilai *k* dan *b* apabila seluruh persamaan pada *subtree* digabung.

4.3.1 Implementasi Fungsi Update

Fungsi *update* memperbarui nilai pada *subtree* dengan *root v* seperti yang dijelaskan pada subbab 3.3.2.

```
1.  void update(){
2.      if(l){
3.          pathk=k*l->pathk%MOD;
4.          pathb=(b+k*l->pathb)%MOD;
```

```
5.      }
6.      else{
7.          pathk=k;
8.          pathb=b;
9.      }
10.     if(r){
11.         pathk=pathk*r->pathk%MOD;
12.         pathb=(r->pathb+r->pathk*pathb)%MOD;
13.     }
14. }
```

Kode Sumber 4.3 Implementasi Update pada Struct Node

4.3.2 Implementasi Splay Tree

Splay Tree digunakan untuk membuat *Auxiliary Tree*. Algoritma s*play tree* dijelaskan pada subbab 2.3.1. Implementasi *splay tree* terdiri dari rotasi kiri (*zig*), rotasi kanan (*zag*), serta *splay*.

```
1.  void zig(){
2.      node *pp = p->p;
3.      if(p->l=r)r->p=p;
4.      r=p;r->p=this;
5.      if(p=pp){
6.          if(p->l==r) p->l=this;
7.          else p->r=this;
8.      }
9.      else{
10.         pathp = r->pathp;
11.         r->pathp = NULL;
12.     }
13.     r->update();
14. }
```

Kode Sumber 4.4 Implementasi Rotasi Kiri pada Splay Tree

```
1.  void zag(){
2.      node *pp = p->p;
3.      if(p->r=l)l->p=p;
4.      l=p;l->p=this;
5.      if(p=pp){
6.          if( p->l==l ) p->l=this;
7.          else p->r=this;
8.      }
9.      else{
10.         pathp = l->pathp;
11.         l->pathp = NULL;
12.     }
13.     l->update();
14. }
```

Kode Sumber 4.5 Implementasi Rotasi Kanan pada Splay Tree

Pada baris 13 dari kedua fungsi dilakukan *update* untuk mempertahankan nilai pada setiap *node* yang turun akibat rotasi.

```
1.  void splay(){
2.      node *pp;
3.      for(; p ;){
4.          if(pp=p->p){
5.              if( pp->l==p ){
6.                  if(p->l==this)p->zig(),zig();
7.                  else zag(),zig();
8.              }
9.              else{
10.                 if(p->r==this)p->zag(),zag();
11.                 else zig(),zag();
12.             }
13.         }
14.         else{
15.             if( p->l==this )zig();
16.             else zag();
```

```
17.         }
18.     }
19.     update();
20. }
```

Kode Sumber 4.6 Implementasi Splay pada Struct Node

Pada akhir splay dilakukan operasi *update* karena *node* yang naik tidak di-*update* nilainya pada zig atau zag.

4.3.3 Implementasi Fungsi Akses

Fungsi akses digunakan untuk membuat jalur dari *node v* sampai menuju *root* pada *represented tree* seperti pada subbab 2.4.1.

```
1.  void access(){
2.      splay();
3.      if( r ){
4.          r->pathp=this;
5.          r->p=NULL;
6.          r=NULL;
7.      }
8.      for(node *pp=pathp ; pp ; pp=pathp ){
9.          pp->splay();
10.         if( pp->r ){
11.             pp->r->pathp=pp;
12.             pp->r->p=NULL;
13.         }
14.         pp->r=this;
15.         p=pp;
16.         pathp=NULL;
17.         zag();
18.     }
19.     update();
20. }
```

Kode Sumber 4.7 Implementasi Fungsi Akses pada Struct Node

Pada akhir fungsi akses dilakukan *update* karena operasi terakhir sebelum keluar dari *for loop* adalah *zag*, yang hanya melakukan *update* pada *node* yang turun, sehingga pada *node* sekarang belum dilakukan *update*.

4.3.4 Implementasi Fungsi Mencari Root

Fungsi mencari *root* mengembalikan pointer menuju *root* pada *represented tree* milik v seperti pada subbab 2.4.2.

```
1.  node* findroot(){
2.      access();
3.      node *ret;
4.      for( ret=this ; ret->l ; ret=ret->l );
5.      ret->splay();
6.      return ret;
7.  }
```

Kode Sumber 4.8 Implementasi Fungsi Mencari Root pada Struct Node

4.3.5 Implementasi Fungsi Penghapusan Edge

Fungsi penghapusan *edge* menghapus hubungan v dengan *parent(v)* pada *represented tree* seperti pada subbab 2.4.3.

```
1.  void cut(){
2.      node *root = findroot();
3.      if( this==root ) cycle=NULL;
4.      else{
5.          access();
6.          l->p = NULL;
7.          l = NULL;
8.          node *temp = root->cycle;
9.          if( temp ){
10.             if( this==temp->findroot() ){
11.                 root->splay();
12.                 root->pathp = root->cycle;
```

```
13.                 root->cycle = NULL;
14.             }
15.         }
16.     }
17. }
```

Kode Sumber 4.9 Implementasi Fungsi Penghapusan Edge pada Struct Node

Karena *tree* yang digunakan bisa memiliki *cycle*, maka apabila *edge* yang dihapus adalah *root* hanya perlu menghapus penunjuk *cycle* pada *root* seperti pada baris 3. Kedua, apabila *edge* yang dihapus membuat sebuah *cycle* putus, maka penunjuk *cycle* diubah menjadi penunjuk *path parent* setelah penghapusan *edge* dilakukan. Hal ini dilakukan dengan cara menelusuri jalur dari penunjuk *cycle*. Apabila jalur terhenti pada v, berarti penghapusan *edge* pada v menyebabkan *cycle* terputus. Implementasi ini terdapat pada baris 9-15.

4.3.6 Implementasi Fungsi Penyisipan Edge

Fungsi penyisipan *edge* menambahkan sebuah *edge* dari v menuju ke n seperti pada subbab 2.4.4.

```
1.  void link(node *n,int k,int b){
2.      this->k=k;
3.      this->b=b;
4.      update();
5.      if( this==n->findroot() ) cycle = n;
6.      else{
7.          access();
8.          n->access();
9.          n->r = this;
10.         p = n;
11.         n->update();
12.     }
13. }
```

Kode Sumber 4.10 Implementasi Fungsi Penyisipan Edge pada Struct Node

Karena *tree* yang digunakan bisa memiliki *cycle*, maka satu hal yang perlu diperhatikan. Apabila v dan n berada pada *auxiliary tree* yang sama, maka tidak dibentuk *edge* baru, melainkan hanya penunjuk *cycle* seperti pada baris 5.

4.3.7 Implementasi Fungsi Getval

Fungsi *getval* menghitung nilai pada persamaan n, seperti pada subbab 3.3.3. Nilai *kAll* dan *bAll* adalah *k1* dan *b1*, sedangkan *kCycle* dan *bCycle* adalah *k2* dan *b2*.

```
1.  int getVal(){
2.      node *root = findroot();
3.      if( !root->no ) return -1;
4.      int k1=root->pathk;
5.      int b1=root->pathb;
6.      root->cycle->access();
7.      int k2=root->cycle->pathk;
8.      int b2=root->cycle->pathb;
9.      if( k2==1 ){
10.         if( b2 ) return -1;
11.         else if( k1 ) return -2;
12.     }
13.     k2 = (1-k2+MOD)%MOD;
14.     int x = inv[k2]*b2%MOD;
15.     return (k1*x+b1)%MOD;
16. }
17. };
```

Kode Sumber 4.11 Implementasi Fungsi Getval pada Struct Node

4.4 Implementasi Fungsi Main

Fungsi *main* diimplementasikan sesuai *pseudocode* pada subbab 3.1. Apabila ONLINE_JUDGE terdefinisi, getchar dan putchar disubstitusikan ke versi lebih cepat. Didefinisikan makro *scan* sebagai metode *input* yang lebih cepat dari *scanf*.

```
1.  #ifdef ONLINE_JUDGE
2.  #define getchar getchar_unlocked
3.  #define putchar putchar_unlocked
4.  #endif
5.  #define scan(a) do c=getchar();while(c<48); \
6.      for(a=0;c>=48;c=getchar())a=a*10+c-48;
7.  int main(){
8.      int n,i,j,k,b;
9.      node tree[30003];
10.     char c,cmd;
11.     pre();
12.     scanf("%d",&n);
13.     for( i=1 ; i<=n ; i++ ){
14.         tree[i].no=i;
15.         scan(k);scan(j);scan(b);
16.         tree[i].link(&tree[j],k,b);
17.     }
18.     for( scanf("%d\n",&n) ; n-- ;){
19.         scan(cmd);
20.         if( cmd=='A'-48 ){
21.             scan(i);
22.             printf("%d\n",tree[i].getVal());
23.         }
24.         else{
25.             scan(i);scan(k);scan(j);scan(b);
26.             tree[i].cut();
27.             tree[i].link(&tree[j],k,b);
28.         }
29.     }
30.     return 0;
31. }
```

Kode Sumber 4.12 Implementasi Fungsi Main

BAB V
UJI COBA DAN EVALUASI

Pada bab ini penulis menjelaskan tentang uji coba dan evaluasi dari implementasi yang telah dilakukan.

5.1 Lingkungan Uji Coba

Lingkungan uji coba yang digunakan sebagai berikut:

1. Perangkat Keras
   Processor Intel® Core™ i7-4700MQ CPU @ 2.40GHz
   RAM 15.9 GB
   64-bit Operating System, x64-based processor
2. Perangkat Lunak
   Operating System Windows 8.1 Single Language
   Integrated Development Environment Orwell Dev-C++ 5.6.3

5.2 Skenario Uji Coba

Pada subbab ini dijelaskan skenario uji coba yang dilakukan.

5.2.1 Uji Coba Kebenaran

Uji coba kebenaran dilakukan dengan mengirim kode sumber implementasi ke situs SPOJ. Permasalahan yang diselesaikan adalah *Dynamic Congruence Equation System* seperti yang dijelaskan pada subbab 2.7. Setelah kode sumber implementasi dikirimkan ke situs SPOJ, dapat dilihat umpan balik sistem pada situs SPOJ seperti yang dijelaskan pada subbab 2.6. Hasil uji coba pada situs SPOJ ditunjukkan pada Gambar 5.1.

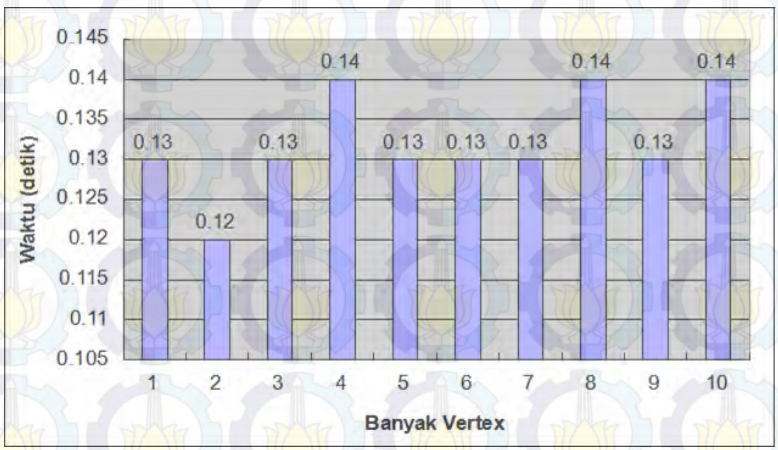
Gambar 5.1 Hasil Uji Coba pada Situs SPOJ

Dari hasil uji coba yang telah dilakukan, kode sumber program yang mendapat umpan balik ***Accepted***. Waktu yang dibutuhkan

program adalah 0.12 detik dan memori yang dibutuhkan program adalah 3.7 MB. Setelah itu dilakukan pengiriman kode sumber implementasi sebanyak 10 kali untuk melihat variasi waktu dan memori yang dibutuhkan program. Hasil uji coba sebanyak 10 kali ditunjukkan dalam Tabel 5.1, Gambar 5.2 dan Gambar 5.3.

Tabel 5.1 Hasil Uji Coba pada Situs SPOJ Sebanyak 10 Kali

No	Hasil	Waktu (detik)	Memori (MB)
1	Accepted	0.13	3.7
2	Accepted	0.12	3.7
3	Accepted	0.13	3.7
4	Accepted	0.14	3.7
5	Accepted	0.13	3.7
6	Accepted	0.13	3.7
7	Accepted	0.13	3.7
8	Accepted	0.14	3.7
9	Accepted	0.13	3.7
10	Accepted	0.14	3.7



Gambar 5.2 Grafik Hasil Uji Coba pada Situs SPOJ Sebanyak 10 Kali

ID		DATE	PROBLEM	RESULT	TIME	MEM
14359287		2015-05-31 03:29:51	Dynamic Congruence Equation System	accepted edit ideone it	0.14	3.7M
14359286		2015-05-31 03:29:50	Dynamic Congruence Equation System	accepted edit ideone it	0.13	3.7M
14359285		2015-05-31 03:29:34	Dynamic Congruence Equation System	accepted edit ideone it	0.14	3.7M
14359284		2015-05-31 03:29:33	Dynamic Congruence Equation System	accepted edit ideone it	0.13	3.7M
14359283		2015-05-31 03:29:33	Dynamic Congruence Equation System	accepted edit ideone it	0.13	3.7M
14359282		2015-05-31 03:29:04	Dynamic Congruence Equation System	accepted edit ideone it	0.13	3.7M
14359281		2015-05-31 03:29:04	Dynamic Congruence Equation System	accepted edit ideone it	0.14	3.7M
14359280		2015-05-31 03:29:03	Dynamic Congruence Equation System	accepted edit ideone it	0.13	3.7M
14359279		2015-05-31 03:28:53	Dynamic Congruence Equation System	accepted edit ideone it	0.12	3.7M
14359276		2015-05-31 03:27:59	Dynamic Congruence Equation System	accepted edit ideone it	0.13	3.7M

Gambar 5.3 Hasil Uji Coba pada Situs SPOJ Sebanyak 10 Kali

Dari hasil pengiriman kode sumber sebanyak 10 kali, didapat waktu rata-rata program yaitu 0.132 detik dan memori rata-rata yang dibutuhkan program yaitu 3.7 MB.

5.2.2 Uji Coba Kinerja

Untuk n *vertex* dan k operasi, algoritma *link cut tree* memiliki kompleksitas waktu *amortized O(k log n)*. Uji coba kinerja dilakukan dengan membuat persamaan dan operasi secara acak. Banyak operasi kueri dan perubahan dibuat hampir sama, dengan urutan acak. Setiap uji coba dilakukan dengan 50 kasus uji kemudian diambil waktu rata-ratanya.

Setiap nilai acak yang dibuat memiliki distribusi uniform dengan batasan yang sama dengan batasan masalah pada subbab 2.7, yaitu sebagai berikut:

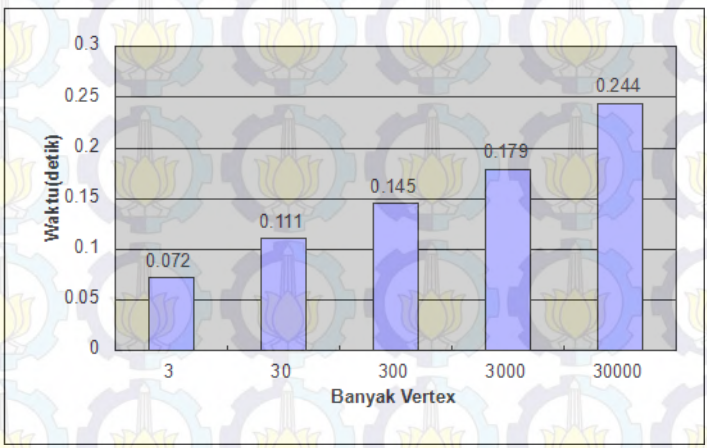
1. $1 \leq N \leq 30000$.
2. $1 \leq Q \leq 100000$.
3. $1 \leq i,p \leq 30000$.
4. $1 \leq k,b \leq 10007$.

5.2.2.1 Pengaruh Banyak Vertex Terhadap Waktu

Banyak *vertex* bervariasi antara 3 sampai 30000 dengan rentang logaritmik berbasis 10. Banyak operasi dibuat tetap yaitu 100000. Hasil uji coba ditunjukkan dalam Tabel 5.2 dan Gambar 5.3.

Tabel 5.2 Hasil Uji Coba Pengaruh Banyak Vertex Terhadap Waktu Eksekusi Program

No	Banyak Vertex	Waktu (detik)
1	3	0.072
2	30	0.111
3	300	0.145
4	3000	0.179
5	30000	0.244



Gambar 5.4 Grafik Hasil Uji Coba Pengaruh Banyak Vertex Terhadap Waktu Eksekusi Program

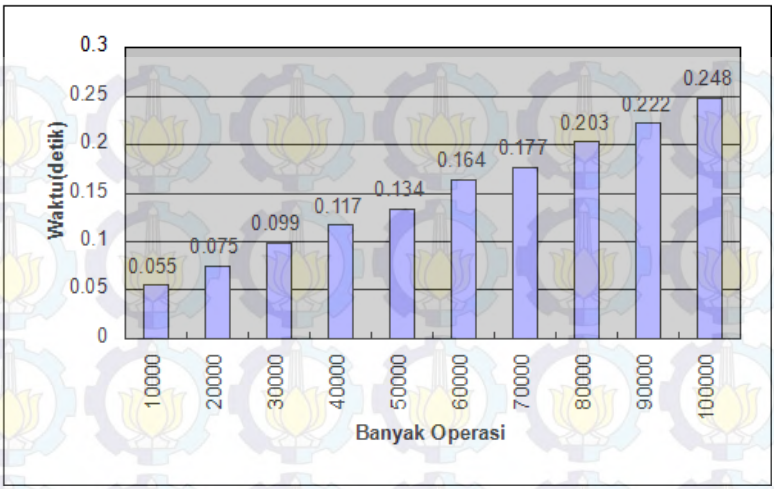
Dari hasil uji coba yang telah dilakukan, dapat dilihat bahwa pertumbuhan waktu yang dibutuhkan program linier seiring dengan pertumbuhan *vertex* yang logaritmik. Hal tersebut membuktikan bahwa banyak *vertex* memiliki pengaruh secara logaritmik terhadap kompleksitas waktu program. Implementasi yang telah dilakukan sesuai dengan kompleksitas waktu dari algoritma *link cut tree* yang dipengaruhi banyak *vertex* secara logaritmik, yaitu *O(k log n)*.

5.2.2.2 Pengaruh Banyak Operasi Terhadap Waktu

Banyak *vertex* dibuat tetap yaitu 30000. Banyak operasi bervariasi antara 10000 sampai 100000 dengan rentang 10000. Hasil uji coba ditunjukkan dalam Tabel 5.3 dan Gambar 5.4.

Tabel 5.3 Hasil Uji Coba Pengaruh Banyak Operasi Terhadap Waktu Eksekusi Program

No	Banyak Operasi	Waktu (detik)
1	10000	0.055
2	20000	0.075
3	30000	0.099
4	40000	0.117
5	50000	0.134
6	60000	0.164
7	70000	0.177
8	80000	0.203
9	90000	0.222
10	100000	0.248



Gambar 5.5 Grafik Hasil Uji Coba Pengaruh Banyak Operasi Terhadap Waktu Eksekusi Program

Dari hasil uji coba yang telah dilakukan, dapat dilihat bahwa pertumbuhan waktu yang dibutuhkan program linier seiring dengan pertumbuhan banyak operasi. Hal tersebut membuktikan bahwa pertumbuhan banyak operasi memiliki pengaruh secara linear terhadap kompleksitas waktu program. Implementasi yang telah dilakukan sesuai dengan kompleksitas waktu dari algoritma *link cut tree* yang dipengaruhi banyak operasi secara linier, yaitu *O(k log n)*.

BAB VI
KESIMPULAN DAN SARAN

Pada bab ini penulis menjelaskan tentang kesimpulan dari hasil uji coba yang telah dilakukan dan saran mengenai hal-hal yang masih bisa dikembangkan.

6.1 Kesimpulan

Dari hasil uji coba yang telah dilakukan terhadap implementasi solusi untuk permasalahan sistem persamaan kongruen yang dinamis dapat diambil kesimpulan sebagai berikut:

1. Sistem persamaan kongruen yang dinamis dapat dimodelkan menjadi struktur data *link cut tree* dengan algoritma *preprocess, update,* dan *getval*.
2. Implementasi *link cut tree* yang telah dilakukan mampu menyelesaikan permasalahan sistem persamaan kongruen yang dinamis dengan benar.
3. Waktu yang dibutuhkan oleh *link cut tree* untuk menyelesaikan permasalahan sistem persamaan kongruen yang dinamis dipengaruhi secara logaritmik terhadap banyak persamaan.
4. Waktu yang dibutuhkan oleh *link cut tree* untuk menyelesaikan permasalahan sistem persamaan kongruen yang dinamis dipengaruhi secara linier terhadap banyak operasi.
5. Algoritma hasil implementasi memiliki kompleksitas waktu *O(k log n)* untuk menyelesaikan permasalahan sistem persamaan kongruen yang dinamis

6.2 Saran

Implementasi pada *link cut tree* yang telah dilakukan menggunakan *splay tree* sebagai struktur data pada *auxiliary tree*. Pengembangan implementasi bisa dilakukan dengan mengganti *splay tree* dengan struktur data lain yang mendukung operasi yang dibutuhkan.

DAFTAR PUSTAKA

[1] Sleator, Daniel D, and Robert Endre Tarjan. "A data structure for dynamic trees." Proceedings of the thirteenth annual ACM symposium on Theory of computing 11 May. 1981: 114-122.

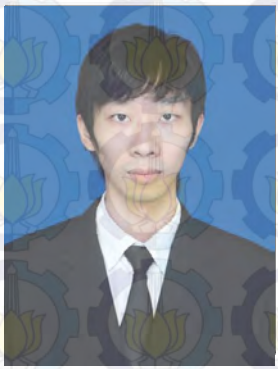
[2] Camil Demetrescu, Irene Finocchi, and Giuseppe F. Italiano. “Dynamic graphs”. 2001.

[3] Sleator, Daniel D.; Tarjan, Robert E. (1985), "Self-Adjusting Binary Search Trees" (PDF), Journal of the ACM (Association for Computing Machinery) 32 (3): 652–686

[4] “SPOJ.com - Problem DCES,” [Online]. Available: http://www.spoj.com/problems/DCES/

[5] “Lecture 19 in 6.851: Advanced Data Structures (Spring’12),” [Online]. Available: https://courses.csail.mit.edu/6.851/spring12/lectures/L19.html

BIODATA PENULIS



Andy William dilahirkan di Surabaya pada tanggal 23 Maret 1993, anak kedua dari 3 bersaudara. Penulis telah menempuh pendidikan formal yaitu di TK Petra 10, SD Petra 10, SMP Petra 1, dan SMA Petra 1. Setelah lulus dari SMA tahun 2011, penulis mengikuti SNMPTN dan diterima di Jurusan Teknik Informatika FTIf-ITS pada tahun 2011 dan terdaftar dengan NRP. 5111100095.

Ketika masih menempuh perkuliahan di Jurusan Teknik Informatika FTIf-ITS, penulis sering menjadi finalis kompetisi pemrograman tingkat nasional yang diselenggarakan di Institut Teknologi Bandung (ITB), Universitas Indonesia (UI), dan Bina Nusantara University (BINUS). Penulis pernah menjadi Juara II (2014) pada kompetisi Gemastik Kategori Programming. Selain itu sejak tahun 2012 hingga saat ini, penulis juga aktif sebagai anggota Himpunan Mahasiswa Teknik Computer-Informatika (HMTC) ITS. Penulis pernah menjadi wakil koordinator (2012) pada National Logic Competition (NLC) Schematics ITS. Selain itu penulis juga pernah menjadi asisten mata kuliah Pemrograman Terstruktur (2012) dan Algoritma dan Struktur Data (2013) di Jurusan Teknik Informatika FTIF-ITS, serta asisten pada Pelatnas 2 dan Pelatnas 3 TOKI (2014).

Perancangan dan Analisis Algoritma Graph Dinamis pada Penyelesaian Permasalahan Sistem Persamaan Kongruen yang Dinamis

Andy William, Ahmad Saikhu, Rully Soelaiman
Jurusan Teknik Informatika, Fakultas Teknologi Informasi, Institut Teknologi Sepuluh Nopember (ITS)
Jl. Arief Rahman Hakim, Surabaya 60111 Indonesia
e-mail: andy11@mhs.if.its.ac.id, saikhu@its-sby.edu, rully@is.its.ac.id

***Abstrak*—Persamaan kongruen merupakan dasar dari banyak teorema pada ilmu matematika. Kumpulan persamaan kongruen yang saling merujuk satu sama lain disebut sistem persamaan kongruen. Solusi umum untuk permasalahan sistem persamaan kongruen tidak cukup, karena pada kenyataannya seringkali terjadi perubahan pada sistem persamaan yang sedang diteliti. Sistem persamaan kongruen yang dapat berubah-ubah disebut sistem persamaan kongruen dinamis. Pada permasalahan sistem kongruen dinamis, hasil persamaan harus dapat dihitung tanpa mengulang dari awal jika terjadi modifikasi pada Persamaan kongruen selalu mengacu persamaan lain, sehingga sistem persamaan dapat dimodelkan menjadi graph. Perubahan persamaan dapat dilambangkan sebagai penghapusan edge dan penyisipan edge pada graph. Graph ini dinamakan link cut tree. Pada penelitian ini diimplementasikan solusi untuk permasalahan sistem persamaan kongruen dinamis dengan menggunakan graph dinamis, yaitu link cut tree. Solusi tersebut dapat menemukan hasil atau mengubah setiap persamaan kongruen pada sistem persamaan kongruen dinamis, serta mempunyai kompleksitas space linear dan kompleksitas time O(log n) untuk setiap operasi.**

***Kata Kunci*— Graph Dinamis, Penghapusan Edge, Penyisipan Edge, Sistem Persamaan Kongruen Dinamis.**

I. PENDAHULUAN

Persamaan kongruen merupakan dasar dari banyak teorema pada ilmu matematika. Kumpulan persamaan kongruen yang saling merujuk satu sama lain disebut sistem persamaan kongruen. Solusi umum untuk permasalahan sistem persamaan kongruen tidak cukup, karena pada kenyataannya seringkali terjadi perubahan pada sistem persamaan yang sedang diteliti. Hal tersebut memunculkan permasalahan sistem persamaan kongruen yang dinamis. Permasalahan yang terjadi adalah bagaimana solusi yang efisien untuk mempertahankan kondisi sistem jika terjadi modifikasi pada persamaan, termasuk penambahan atau perubahan persamaan. Solusi yang paling mudah adalah melakukan komputasi ulang dari nol setiap terjadi modifikasi pada sistem, tetapi hal tersebut tentu saja tidak efisien.

Pada permasalahan tersebut, terdapat N buah persamaan kongruen dalam bentuk $x[i] = k_i \times x[p_i] + b_i \pmod{10007}$. Kemudian, ada 2 jenis operasi yang dapat dilakukan, yaitu mencari hasil dari x[i] untuk suatu i, atau mengubah nilai persamaan x[i] untuk suatu i. Setiap persamaan $x[i]$ dijadikan sebuah vertex i. Kemudian, vertex i dihubungkan ke vertex $x[p_i]$ dengan sebuah *edge*. Karena persamaan dapat berubah-ubah, maka *graph* juga dapat berubah-ubah sehingga menjadi *graph* dinamis.

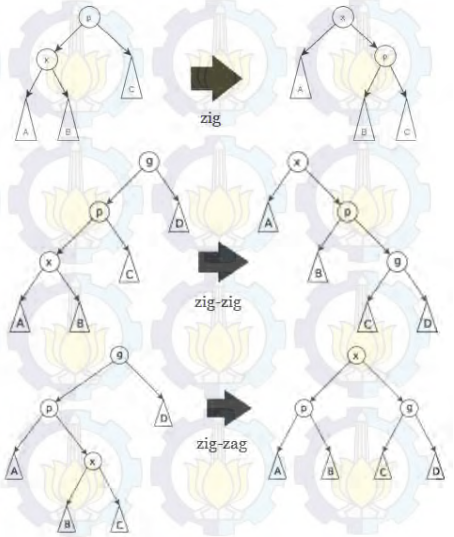
Pada penelitian ini diimplementasikan solusi untuk permasalahan sistem persamaan kongruen yang dinamis dengan menggunakan *graph* dinamis[*1*]. Solusi tersebut mempunyai kompleksitas waktu *O(log n)* untuk setiap kueri pengambilan hasil dan perubahan persamaan.

Untuk melakukan operasi pada sistem persamaan diperlukan informasi dari persamaan-persamaan yang terhubung baik secara langsung maupun rekursif. Diperlukan struktur data yang efisien dan mampu menyimpan atau memodifikasi properti dari semua *vertex* dalam suatu jalur. Struktur data *link cut tree* dipilih karena karakter utamanya yang melakukan operasi pada tree tidak berdasarkan *subtree*, melainkan berdasarkan jalur.

II. TINJAUAN PUSTAKA

A. Splay Tree

Splay tree adalah sebuah *binary search tree* yang strukturnya cenderung seimbang, berarti *vertex* terjauh dari *root* diusahakan sedekat mungkin, sehingga setiap *vertex* dapat diakses dengan waktu yang relatif sama, yaitu *O(log N)*[*3*]. Apabila *vertex x* diakses pada *splay tree S*, maka dilakukan operasi *splay(x)*. Pada *splay(x),* struktur S diubah sehingga *x* menjadi root dari *S*.

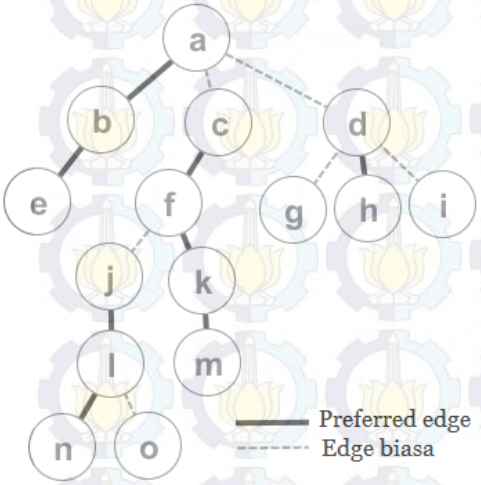


Gambar 1. Ilustrasi kemungkinan dari setiap iterasi splay

Setiap iterasi pada operasi *splay(x)* tergantung pada *x*, *parent(v)* yaitu p, serta *parent(p)* yaitu g. Apabila *p* adalah root dari *splay tree*, maka *tree* dirotasi pada *edge(p,x)*. Operasi ini dinamakan *zig*. Apabila tidak, serta jalur dari *x* ke *g* tidak berbelok, maka *edge(g,p)* dirotasi kemudian *edge(p,x)* dirotasi. Operasi ini dinamakan *zig-zig*. Selain itu, maka *edge(p,x)* dirotasi kemudian *edge(g,x)* dirotasi. Operasi ini dinamakan *zig-zag*. Operasi *splay(v)* melakukan hal ini secara berulang sampai *x* menjadi root dari *splay tree*. Ilustrasi operasi splay ditunjukkan pada Gambar 1.

B. Link Cut Tree

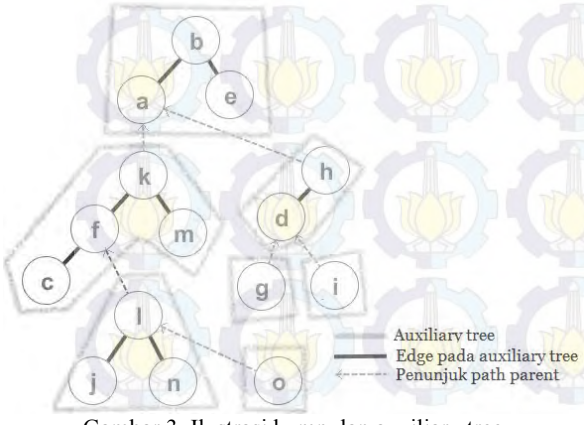
Link cut tree adalah struktur data untuk merepresentasikan kumpulan *tree*[*2*]. Setiap *tree* dipisah menjadi jalur atau *path* yang linier. Kumpulan jalur tersebut dinamakan *represented tree*, diilustrasikan pada Gambar 2.



Gambar 2. Ilustrasi represented tree

Ketika sebuah *vertex* v diakses pada *represented tree RT,* maka jalur dari v menuju *root* menjadi *preferred path*. Setiap *vertex* hanya dapat terletak pada satu *preferred path*. *Edge* yang menghubungkan *preferred path* disebut *preferred edge*.

Setiap *path p* pada *RT* direpresentasikan oleh *auxiliary tree AT* dalam bentuk *splay tree*. Untuk setiap *AT*, *vertex* yang lebih dekat dengan *root* pada *RT* memiliki nilai yang lebih kecil, sehingga *AT* terurut berdasarkan kedalaman atau *depth* pada *RT*. Ilustrasi *auxiliary tree* ditunjukkan pada Gambar 3.

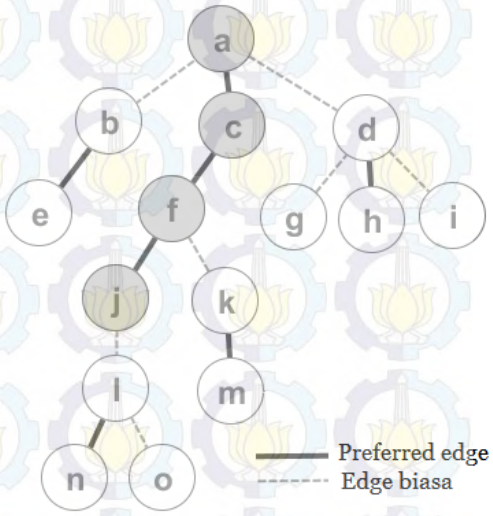


Gambar 3. Ilustrasi kumpulan auxiliary tree

Edge yang terhubung pada *RT* sebagai *edge* biasa akan terhubung oleh penunjuk *path parent* pada *AT*. Namun, penunjuk *path parent* selalu terletak pada root dari *AT*.

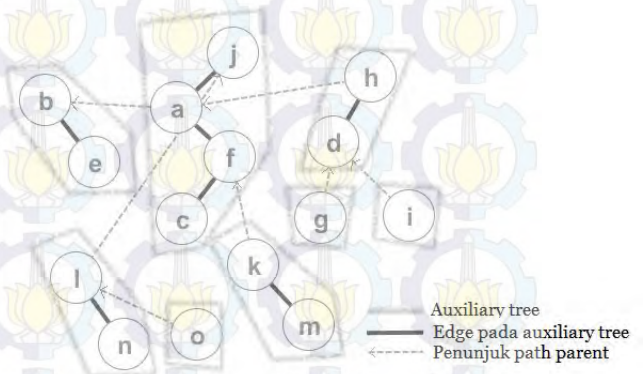
B. 1. Algoritma Akses

Operasi *access(v)* membuat *preferred path* dari *vertex* v menuju *root(RT)*. Setelah *access(v)* dilakukan, *vertex* v menjadi ujung dari *path(v)*. Jalur dari v menuju *root(RT)* akan sepenuhnya terdiri dari *preferred edge*. Ilustrasi *represented tree* setelah melakukan *access(j)* ditunjukkan pada Gambar 4.



Gambar 4. Ilustrasi represented tree setelah melakukan access(j)

Dalam *auxiliary tree*, operasi ini dilakukan dengan cara melakukan *splay(v)*. Setelah itu *child* kanan dipisah menjadi *auxiliary tree* yang baru, dengan penunjuk *path parent* ke *v*. Operasi dilanjutkan dengan menggabungkan *auxiliary tree* dari *v* dengan *auxiliary tree* dari penunjuk *path parent*. Untuk menggabungkan kedua *auxiliary tree*, maka pada *auxiliary tree* milik *path parent* juga harus dilakukan operasi seperti pada *v*. Selanjutnya, jadikan *v* sebagai *child* kanan dari *path parent*, dan lakukan *splay(v)*. Operasi ini dilakukan terus sampai *auxiliary tree* milik *v* tidak memiliki *path parent*. Ilustrasi akses ditunjukkan pada Gambar 5.



Gambar 5. Ilustrasi auxiliary tree setelah melakukan access(j)

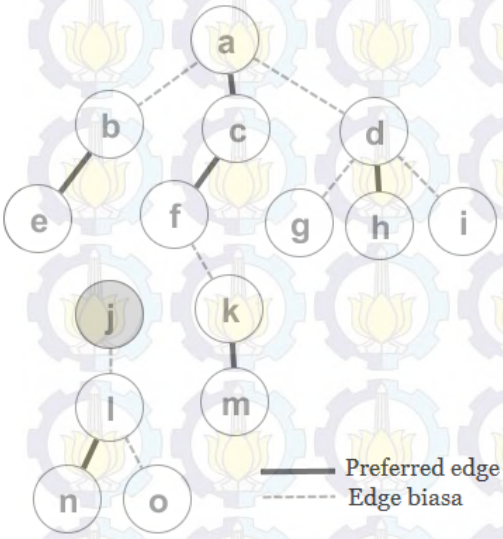
B. 2. Algoritma Mencari Root

Untuk mencari *root* dari *v*, pertama lakukan *access(v)*, kemudian hasilnya adalah *vertex* dengan nilai terkecil atau

paling kiri pada *auxiliary tree* dari v. Setelah itu lakukan *splay* pada *root* tersebut.

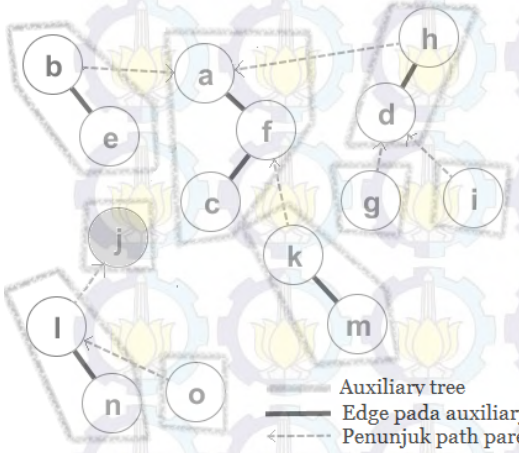
B. 3. Algoritma Penghapusan Edge

Operasi *cut(v)* menghapus *edge v* dengan *parent*-nya. Setelah operasi penghapusan *edge* dilakukan, *tree* akan terpisah menjadi dua, dengan *v* menjadi *root* dari *tree* yang baru. Ilustrasi *represented tree* setelah melakukan *cut(j)* ditunjukkan pada Gambar 6.



Gambar 6. Ilustrasi represented tree setelah melakukan cut(j)

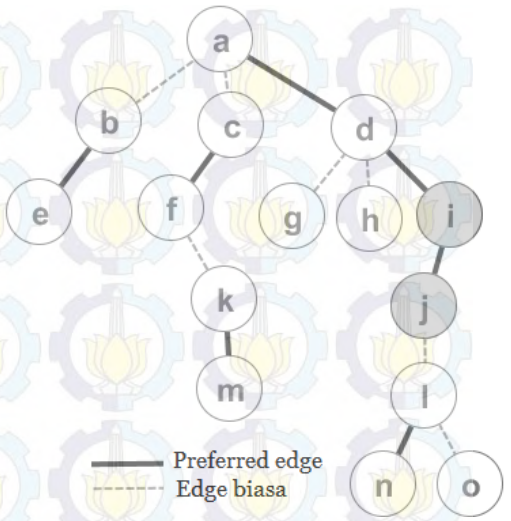
Operasi *cut(v)* dilakukan dengan melakukan operasi *access(v)*. Kemudian hapus hubungan v dengan *child* kirinya, tanpa menambahkan penunjuk *path parent*. Ilustrasi penghapusan *edge* ditunjukkan pada Gambar 7.



Gambar 7. Ilustrasi auxiliary tree setelah melakukan cut(j)

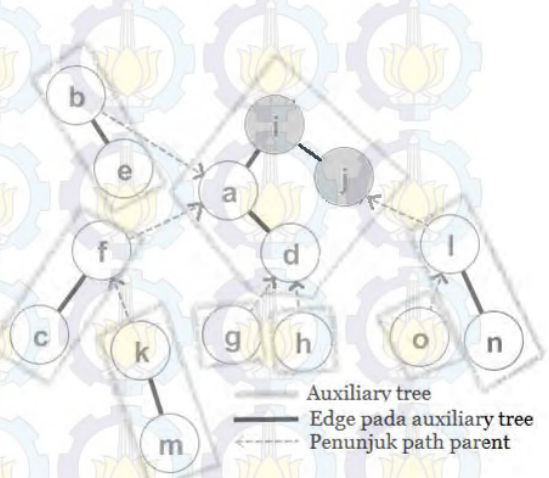
B. 4. Algoritma Penyisipan Edge

Operasi *link(u,v)* menghubungkan *vertex* u dan v dari dua *represented tree* yang berbeda menjadi satu, dengan syarat properti *tree* harus dipertahankan. Ilustrasi *represented tree* setelah melakukan *link(j,i)* ditunjukkan pada Gambar 8.



Gambar 8. Ilustrasi represented tree setelah melakukan link(j,i)

Operasi *link(u,v)* dilakukan dengan cara melakukan *access(u)* dan *access(v)*, kemudian gabungkan jadikan *u* sebagai *child* kanan dari *v*. Ilustrasi penyisipan *edge* ditunjukkan pada Gambar 9.



Gambar 9. Ilustrasi auxiliary tree setelah melakukan link(j,i)

C. Metode Invers Modular

Inverse modular dari sebuah bilangan a modulus m adalah x dimana $ax \equiv 1 \pmod{m}$. Dapat dikatakan bahwa $a^{-1} \equiv x \pmod{m}$. Sebagai contoh, kita ingin mencari hasil dari $x \equiv 3^{-1} \pmod{11}$. Hal ini sama dengan mencari hasil dari $3x \equiv 1 \pmod{11}$. Satu-satunya hasil yang juga memenuhi $1 \leq x \leq 11$ adalah 4. Berarti, pembagian dengan 3 pada sebuah persamaan dengan modulus 11 dapat diubah menjadi perkalian dengan 4. Contoh kasusnya adalah $12/3 \equiv 12 \times 4 \pmod{11}$, kedua sisi sama-sama menghasilkan hasil 4 setelah dimodulus 11.

Karena sifat modulus yang distributif pada perkalian, angka besar yang terdiri dari banyak perkalian dapat dihitung secara iteratif per suku. Karena pembagian dapat diubah menjadi perkalian dengan invers modular, maka angka besar yang mengandung pembagian juga dapat dihitung.

Pembagian dengan a sama dengan perkalian dengan invers a, maka $a^{-n} \equiv x^n \pmod{m}$. Metode paling cepat untuk melakukan perhitungan semua invers modular dari 1 sampai m-1 adalah

mencari nilai n sehingga untuk n=1 sampai m-1 nilai a^n berbeda semua. Untuk modulus 10007, nilai n yang cocok adalah 5, dengan nilai x 4003.

III. METODOLOGI PERANCANGAN

Program dirancang berdasarkan tinjauan pustaka pada Bab II dengan beberapa modifikasi berikut.

A. *Splay Tree*

Pada *splay tree*, seluruh *vertex* yang berada dalam *subtree s* dapat diubah nilainya tanpa mengubah satu persatu *vertex*. Perubahan yang ingin dilakukan hanya disimpan pada *root* dari *s*. Apabila elemen didalam *s* ingin diakses barulah perubahan tersebut diturunkan ke *child* nya.

Hal ini perlu dilakukan karena setiap *vertex* tidak hanya menyimpan nilai k_i dan b_i. Setiap *vertex i* juga perlu menyimpan nilai k_i dan b_i dari gabungan seluruh persamaan pada *subtree* dengan root *i*. Nilai gabungan adalah nilai *k* dan *b* apabila pada seluruh persamaan yang berhubungan dilakukan substitusi sehingga menjadi satu persamaan.

B. *Link Cut Tree*

Karena setiap persamaan memiliki satu *edge* keluar, maka *root* juga terhubung ke *vertex* lain sehingga menyebabkan terjadinya *cycle*. Namun, *tree* tidak boleh memiliki *cycle*. Untuk mengakali hal tersebut, *vertex* yang dituju dari *root* hanya dilambangkan sebagai sebuah penunjuk *cycle*.

Apabila ada operasi penghapusan *edge* pada *cycle* tersebut, maka penunjuk *cycle* dapat diubah menjadi *edge* biasa. Apabila ada operasi penyisipan *edge* pada *tree* yang sama, akan terbentuk *cycle* sehingga penyisipan *edge* tersebut hanya terdiri dari penambahan penunjuk *cycle*.

C. *Menghitung Hasil Persamaan*

Sebuah persamaan tidak akan memiliki jawaban apabila jika ditelusuri secara rekurens pada akhirnya akan merujuk pada persamaan yang belum terdifinisi. Sebuah persamaan dapat dicari nilai pastinya apabila jika ditelusuri secara rekurens akan merujuk kembali ke persamaan itu sendiri, sehingga bentuk persamaan setelah dilakukan substitusi sampai kembali ke dirinya sendiri menjadi $x[i] = pathk \times x[p_i] + pathb \pmod{10007}$. Nilai $x[i]$ kemudian dapat dicari dengan menyelesaikan persamaan tersebut, yaitu $x[i](1-pathk) = pathb \pmod{10007}$. Perlu diperhatikan apabila nilai $pathk=1$, maka nilai *pathb* yang memenuhi hanya $pathb=0$(nilai $x[i]$ bisa diisi berapapun) dan persamaan tidak memiliki hasil apabila $pathb \neq 0$.

Apabila sebuah persamaan *x[i]* memiliki solusi, maka setiap persamaan $x[j] = k_i \times x[p_i] + b_i \pmod{10007}$, atau dengan kata lain persamaan yang merujuk kepada *x[i],* baik secara langsung maupun melalui relasi rekurens, juga akan memiliki hasil.

IV. UJI COBA DAN EVALUASI

A. *Uji Kebenaran*

Uji coba kebenaran dilakukan dengan mengirim kode sumber implementasi ke situs SPOJ. Permasalahan yang diselesaikan adalah *Dynamic Congruence Equation System*[*4*]. Setelah kode sumber implementasi dikirimkan ke situs SPOJ, dapat dilihat umpan balik sistem pada situs SPOJ. Dari hasil uji coba yang telah dilakukan, kode sumber program yang mendapat umpan balik Accepted. Waktu yang dibutuhkan program adalah 0.12 detik dan memori yang dibutuhkan program adalah 3.7 MB.

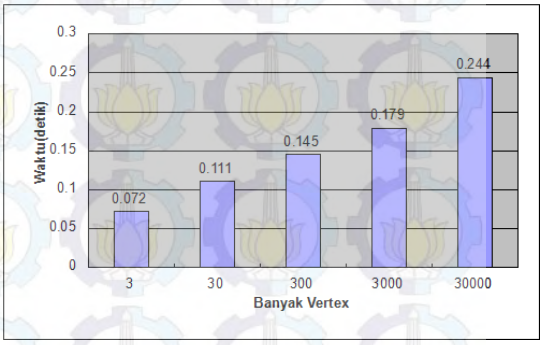
B. *Uji Kinerja*

Uji coba kinerja dilakukan dengan membuat persamaan dan operasi secara acak. Banyak operasi kueri dan perubahan dibuat hampir sama, dengan urutan acak. Setiap nilai acak yang dibuat memiliki distribusi uniform dengan batasan yang sama dengan batasan masalah. Setiap uji coba dilakukan dengan 50 kasus uji kemudian diambil waktu rata-ratanya.

Pada uji coba pertama, banyak *vertex* bervariasi antara 3 sampai 30000 dengan rentang logaritmik berbasis 10. Banyak operasi dibuat tetap yaitu 100000. Hasil uji coba ditunjukkan dalam Tabel 1 dan Gambar 10.

Tabel 1. Uji Coba Pengaruh Banyak Vertex Terhadap Waktu

No	Banyak Vertex	Waktu (detik)
1	3	0.072
2	30	0.111
3	300	0.145
4	3000	0.179
5	30000	0.244



Gambar 10 Grafik Uji Coba Pengaruh Banyak Vertex Terhadap Waktu

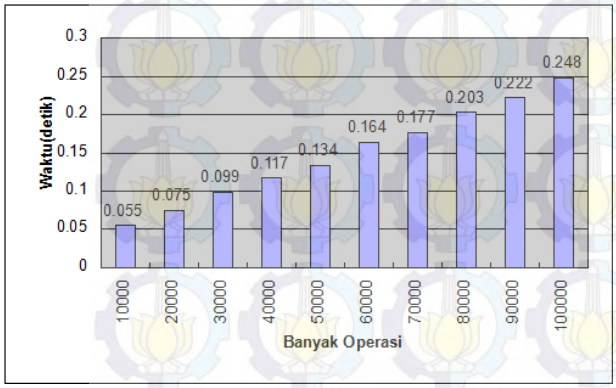
Dari hasil uji coba yang telah dilakukan, pertumbuhan waktu yang dibutuhkan program linier seiring dengan pertumbuhan *vertex* yang logaritmik. Hal tersebut berarti pertumbuhan waktu terhadap banyak *vertex* logaritmik dan implementasi yang telah dilakukan sesuai dengan kompleksitas waktu dari algoritma link cut tree yang dipengaruhi banyak *vertex* secara logaritmik.

Pada uji coba kedua, banyak *vertex* dibuat tetap yaitu 30000. Banyak operasi bervariasi antara 10000 sampai 100000 dengan rentang 10000. Hasil uji coba ditunjukkan dalam Tabel 2 dan Gambar 11

Tabel 2. Uji Coba Pengaruh Banyak Operasi Terhadap Waktu

No	Banyak Operasi	Waktu (detik)
1	10000	0.055
2	20000	0.075
3	30000	0.099
4	40000	0.117
5	50000	0.134

6	60000	0.164
7	70000	0.177
8	80000	0.203
9	90000	0.222
10	100000	0.248



Gambar 11. Grafik Uji Coba Pengaruh Banyak Operasi Terhadap Waktu

Dari hasil uji coba yang telah dilakukan, pertumbuhan waktu yang dibutuhkan program linier seiring dengan pertumbuhan banyak operasi. Hal tersebut berarti pertumbuhan waktu terhadap banyak operasi linier dan implementasi yang telah dilakukan sesuai dengan kompleksitas waktu dari algoritma *link cut tree* yang dipengaruhi banyak operasi secara linier.

V. KESIMPULAN

Dari hasil uji coba yang telah dilakukan terhadap implementasi solusi untuk permasalahan sistem persamaan kongruen yang dinamis dapat diambil kesimpulan sebagai berikut:

1. Implementasi yang telah dilakukan dapat menyelesaikan permasalahan sistem persamaan kongruen yang dinamis dengan benar.
2. Struktur data *link cut tree* mempunyai *amortized time complexity O(log n)* untuk setiap operasi akses, mencari *root*, penghapusan *edge*, penyisipan *edge*, dan *getval*.

UCAPAN TERIMA KASIH

Penulis mengucapkan puji syukur kepada Tuhan Yang Maha Esa yang telah memberikan rahmat dan karunia-Nya, sehingga penulis dapat menyelesaikan penelitian dengan lancar. Penulis juga mengucapkan terima kasih kepada Bapak Ahmad Saikhu dan Bapak Rully Soelaiman yang telah banyak membantu penulis dalam menyelesaikan penelitian ini. Penulis juga mengucapkan terima kasih banyak kepada pihak-pihak lain yang turut membantu penulis untuk menyelesaikan penelitian ini.

DAFTAR PUSTAKA

[1] Sleator, Daniel D, and Robert Endre Tarjan. "A data structure for dynamic trees." Proceedings of the thirteenth annual ACM symposium on Theory of computing 11 May. 1981: 114-122.

[2] Camil Demetrescu, Irene Finocchi, and Giuseppe F. Italiano. “Dynamic graphs”. 2001.

[3] Sleator, Daniel D.; Tarjan, Robert E. (1985), "Self-Adjusting Binary Search Trees" (PDF), Journal of the ACM (Association for Computing Machinery) 32 (3): 652–686

[4] SPOJ-Problem DCES. http://www.spoj.com/problems/DCES

PERANCANGAN DAN ANALISIS ALGORITMA GRAPH DINAMIS PADA PENYELESAIAN PERMASALAHAN SISTEM PERSAMAAN KONGRUEN YANG DINAMIS

Penyusun Tugas Akhir:
Andy William (NRP 5111 100 095)

Dosen Pembimbing:
Ahmad Saikhu, S.Si., M.T.
Rully Soelaiman, S.Kom., M.Kom.

KERANGKA PRESENTASI

- Pendahuluan
  - Latar Belakang
  - Rumusan Masalah
  - Batasan Masalah
  - Tujuan
- Ilustrasi
- Uji Coba dan Evaluasi
- Kesimpulan

LATAR BELAKANG

- Latar belakang Tugas Akhir ini adalah permasalahan Dynamic Congruence Equation System pada situs SPOJ.
- Permasalahan tersebut adalah mencari hasil dari sistem persamaan kongruen. Tetapi, variabel dari setiap persamaan dalam sistem dapat berubah sewaktu-waktu.
- Solusi yang paling mudah adalah melakukan komputasi ulang setiap terjadi modifikasi pada sistem, tetapi hal tersebut tentu saja tidak efisien.

RUMUSAN MASALAH

1. Bagaimana desain algoritma dan struktur data yang efisien untuk permasalahan sistem persamaan kongruen yang dinamis?
2. Bagaimana implementasi algoritma dan struktur data yang efisien berdasarkan desain yang telah dilakukan?
3. Bagaimana uji coba untuk mengetahui kebenaran dan kinerja dari implementasi yang telah dilakukan?

BATASAN MASALAH

1. Implementasi dilakukan dengan bahasa pemrograman C++.
2. Batasan pada permasalahan Dynamic Congruence Equation System pada SPOJ.

TUJUAN

- Mengimplementasikan solusi permasalahan sistem persamaan kongruen dinamis.
- Melakukan uji coba untuk mengetahui kebenaran dan kinerja dari implementasi yang telah dilakukan.

KERANGKA PRESENTASI

- Pendahuluan
- Ilustrasi
  - Ilustrasi Permasalahan
  - Ilustrasi Solusi
- Uji Coba dan Evaluasi
- Kesimpulan

ILUSTRASI PERMASALAHAN

- Permasalahan Dynamic Congruence Equation System pada Sphere Online Judge (SPOJ) (http://www.spoj.com/problems/DCES/)

DCES - Dynamic Congruence Equation System

no tags

Consider the congruence equation system as the following form:

x[1] = k1 x[p1] + b1 (mod 10007)
x[2] = k2 x[p2] + b2 (mod 10007)
...
x[n] = kn x[pn] + bn (mod 10007)

We will ask you to achieve some instructions as the following form:

- A i: Ask the current x[i]'s value. (or "-1" for no solution, "-2" for multiply solution.)
- C i k p b: Modify the ith congruence equation to a new one.

DESKRIPSI PERMASALAHAN

1. Diberikan N buah persamaan kongruen dalam bentuk:
   $x[i] = k_i\, x[p_i] + b_i \pmod{10007}$
2. Ada Q buah operasi yang harus dilakukan pada sistem persamaan tersebut. Operasi yang mungkin dilakukan adalah:
   Diberikan i, keluarkan hasil pada persamaan $x[i]$.
   Diberikan i, k, p, dan b, ubah nilai persamaan $x[i]$ sehingga $x[i] = k\, x[p] + b \pmod{10007}$.

CONTOH MASUKAN DAN KELUARAN

5
2 2 1
2 3 2
2 4 3
2 5 4
2 3 5
5
A 1
A 2
C 5 3 1 1
A 4
A 5

Bentuk Persamaan

Baris pertama masukan berisi sebuah bilangan bulat N

$x[1] = 2\,x[2] + 1$

$x[2] = 2\,x[3] + 2$

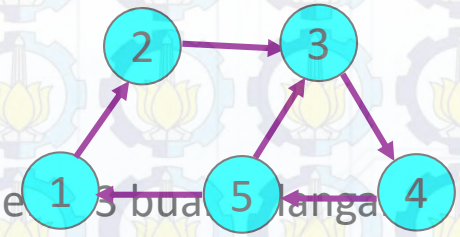
$x[3] = 2\,x[4] + 3$

$x[4] = 2\,x[5] + 4$

$x[5] = 3\,x[1] + 1$

N baris berikutnya masing-masing berisi 3 buah bilangan bulat, yaitu i, pi, dan bi, urutan dari i=1 sampai N

Bentuk Graph



Baris berikutnya berisi sebuah bilangan bulat Q

Q baris berikutnya masing-masing berisi deskripsi operasi yang dilakukan

"A i" artinya keluarkan hasil persamaan $x[i]$

Apabila tidak ada nilai yang memenuhi $x[i]$, keluarkan -1

Apabila ada lebih dari satu nilai yang memenuhi $x[i]$, keluarkan -2

Untuk setiap operasi yang berawalan 'A', keluarkan hasil dari $x[i]$

pehiamaan $x[i] = k\,x[p] + b \pmod{10007}$

Keluaran

4276

7141

BATASAN PERMASALAHAN

- $1 \leq N \leq 30000$.
- $1 \leq Q \leq 100000$.
- Lingkungan penilaian Intel Pentium G860 3GHz.
- Batasan waktu 0.140 detik.
- Batasan memori 1536 MB.

Algoritma Brute Force
= O(N*Q)
= 30000*100000
= 3000000000

Algoritma dengan menggunakan link cut tree:
= O(Q*log(N))
= 100000*15
= 1500000

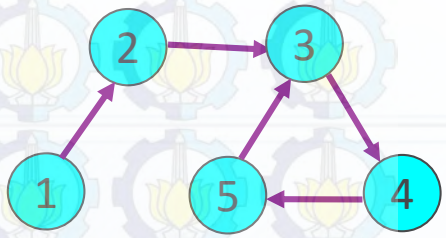
ILUSTRASI SOLUSI

- Persamaan awal diubah ke bentuk represented tree
- Dalam contoh ini vertex 4 dijadikan root
- Tree tidak boleh memiliki cycle, sehingga cycle pada represented tree "dihapus"
- Garis putus-putus warna hijau pada represented tree adalah penanda bahwa terdapat cycle pada graph aslinya

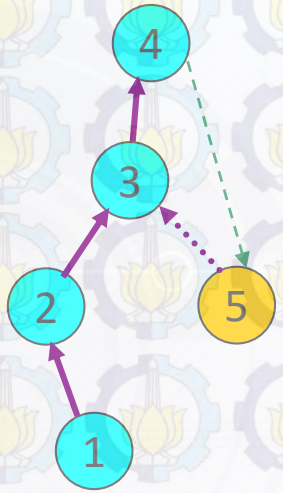
Bentuk Persamaan

$$x[1] = 2\,x[2] + 1$$
$$x[2] = 2\,x[3] + 2$$
$$x[3] = 2\,x[4] + 3$$
$$x[4] = 2\,x[5] + 4$$
$$x[5] = 2\,x[3] + 5$$

Bentuk Graph



Represented Tree



REPRESENTED TREE

- Setiap jalur menyimpan nilai k dan b dari gabungan seluruh persamaan di dalamnya, dinamai pathk dan pathb
- Pada jalur oranye, karena hanya terdiri dari 1 vertex maka nilai pathk dan pathb sama dengan nilai k dan b
- Pada jalur hijau biru, nilai pathk dan pathb dicari menggunakan substitusi dari vertex terdalam, yaitu vertex 1

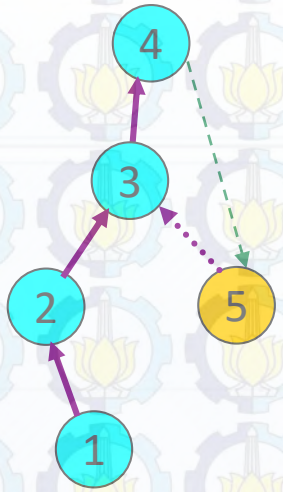
Bentuk Persamaan

$$x[1] = 2\,x[2] + 1$$
$$x[2] = 2\,x[3] + 2$$
$$x[3] = 2\,x[4] + 3$$
$$x[4] = 2\,x[5] + 4$$
$$x[5] = 2\,x[3] + 5$$

Jalur Oranye
pathk=2
pathb=5

Jalur Hijau Biru
pathk=16
pathb=49

Represented Tree

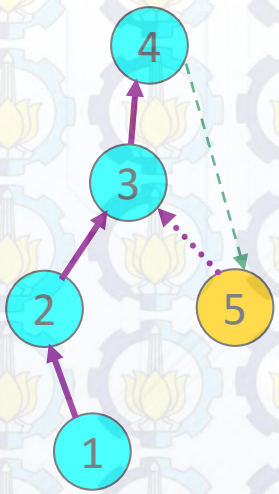


$$x[1] = 2\,x[2] + 1$$
$$x[1] = 4\,x[3] + 3$$
$$x[1] = 8\,x[4] + 7$$
$$x[1] = 16\,x[5] + 49$$

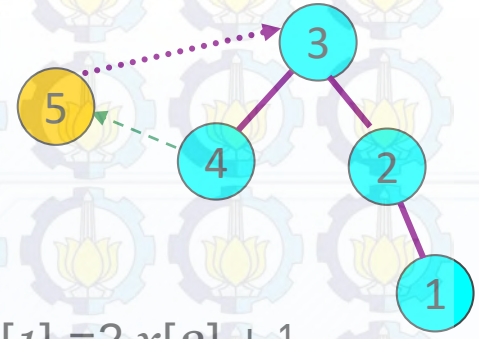
AUXILIARY TREE

- Agar pathk dan pathb dapat diupdate dengan cepat, maka pathk dan pathb disimpan dalam setiap vertex pada auxiliary tree
- Pada auxiliary tree, nilai pathk dan pathb dari suatu vertex v mewakili subtree dengan root v

Represented Tree



Auxiliary Tree



$$x[1] = 2\,x[2] + 1$$
$$x[3] = 4\,x[3] + 5$$
$$x[4] = 2\,x[5] + 4$$
$$x[1] = 16\,x[5] + 49$$
$$x[5] = 2\,x[3] + 5$$

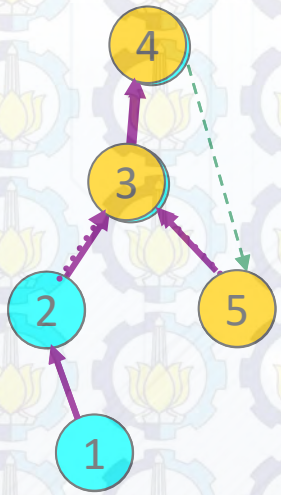
Vertex	1->2	2->3	3->4	4->5	5->3
k	2	2	2	2	2
b	1	2	3	4	5

Path	1->2	1->3	1->5	4->5	5->3
pathk					
pathb					

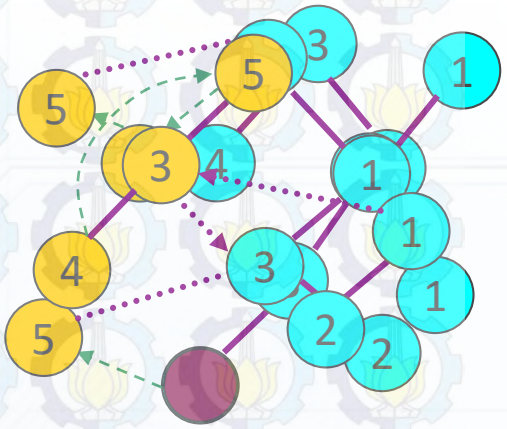
MENCARI NILAI X[1]

- Cari root dari x[1]
  Akses x[1]
  Root adalah vertex paling kiri yaitu x[4]
  Splay x[4]
- Dari pathk dan pathb x[4] didapat:
  x[1] = 16 x[5] + 49 (mod 10007) (1)
- Akses cycle dari root, yaitu x[5]
- Dari pathk dan pathb x[5] didapat:
  x[5] = 8 x[5] + 27 (mod 10007) (2)
- Dari persamaan 2, hitung hasil x[5]:
  x[5] = -27/7 (mod 10007)
  x[5] = -27*7148(mod 10007)
  x[5] = 7144 (mod 10007)
- Lalu hitung x[1] dari persamaan 1:
  x[1] = 16 * 7144 + 49 (mod 10007)
  x[1] = 4276 (mod 10007)

Represented Tree



Auxiliary Tree



Vertex	1->2	2->3	3->4	4->5	5->3
k	2	2	2	2	2
b	1	2	3	4	5

Path	1->3	2->3	3->5	4->5	5->5
pathk	4	2	4	2	8
pathb	5	2	11	4	27

MENGUBAH NILAI X[5]

- **Sesuai contoh input "C 5 3 1 1"**
  x[5] = 3 x[1] + 1 (mod 10007)
- **Cut x[5]**
  Karena cycle terputus, maka jadikan edge cycle menjadi edge asli (path parent)

  Pada auxiliary tree, x[4] di-splay terlebih dahulu karena path parent harus selalu berada di root
- **Cari root dari x[1]**
  Akses x[1]

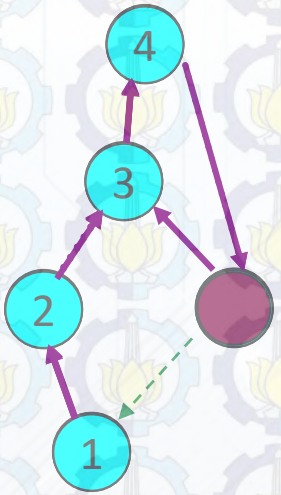
  Pada auxiliary tree, apabila root berubah, path parent menjadi terletak pada root yang baru

  Didapatkan root dari x[1] adalah x[5]
- **Link x[5] ke x[1]**
  x[5] adalah root dari x[1], sehingga pemasangan akan menimbulkan cycle

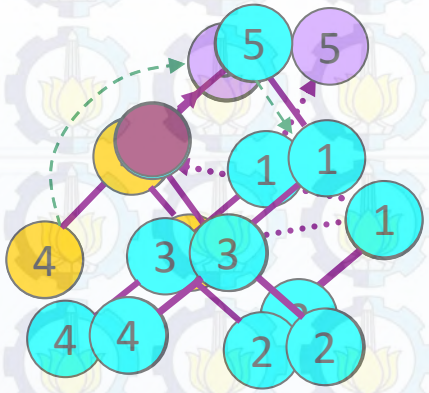
  Tinggal menambahkan pointer cycle

  Perbarui nilai k, b, pathk, dan pathb

Represented Tree



Auxiliary Tree



Vertex	1->2	2->3	3->4	4->5	5->1
k	2	2	2	2	3
b	1	2	3	4	1

Path	1->5	2->3	2->5	4->5	1->1
pathk	16	2	8	2	48
pathb	49	2	24	4	65

KERANGKA PRESENTASI

- Pendahuluan
- Ilustrasi
- Uji Coba dan Evaluasi
  - Uji Coba Kebenaran
  - Uji Coba Kinerja
- Kesimpulan

LINGKUNGAN UJI COBA

1. Perangkat Keras

   Processor Intel® Core™ i7-4700MQ CPU @ 2.40GHz

   RAM 15.9 GB

   64-bit Operating System, x64-based processor

2. Perangkat Lunak

   Operating System Windows 8.1 Single Language

   Integrated Development Environment Orwell Dev-C++ 5.6.3

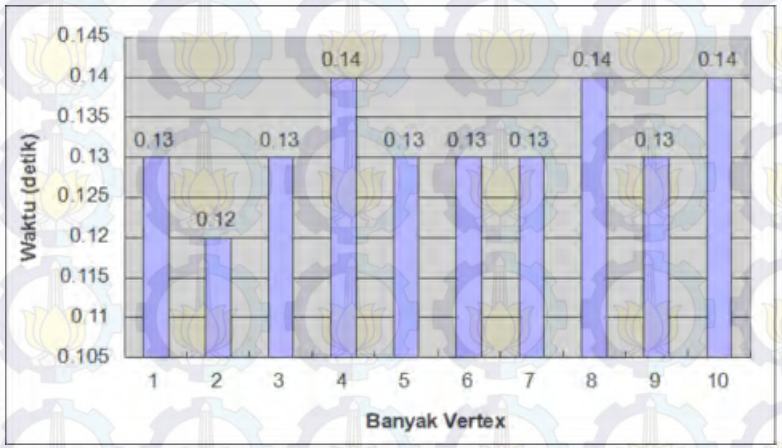
UJI COBA KEBENARAN

- Implementasi yang telah dilakukan program mendapat umpan balik ***Accepted***. Waktu yang dibutuhkan program adalah 0.12 detik dan memori yang dibutuhkan program adalah 3.7 MB.

14359252		2015-05-31 03:08:17	Dynamic Congruence Equation System	accepted edit recheck	0.12	3.7M	C++ 4.3.2

- Waktu yang dibutuhkan program minimum 0.12 detik, maksimum 0.14 detik, dan modus 0.13. Memori yang dibutuhkan program 3.7 MB.

No	Waktu (detik)	Memori (MB)
1	0.13	3.7
2	0.12	3.7
3	0.13	3.7
4	0.14	3.7
5	0.13	3.7
6	0.13	3.7
7	0.13	3.7
8	0.14	3.7
9	0.13	3.7



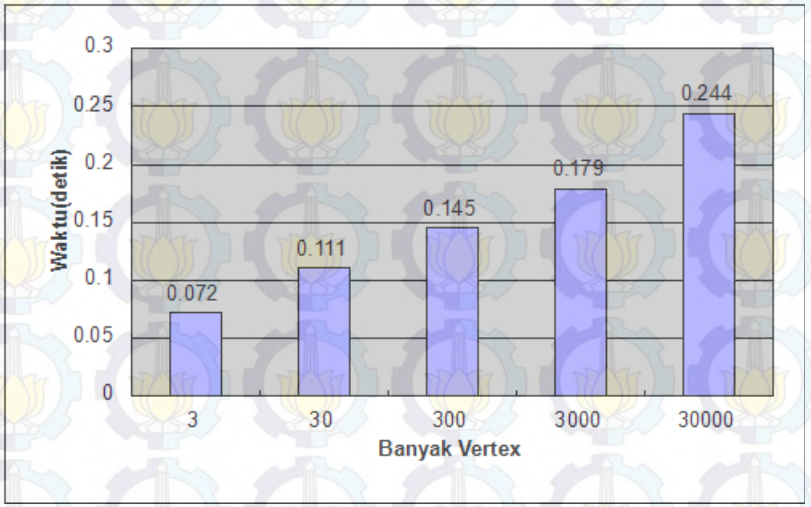
UJI COBA KINERJA

- Uji coba kinerja dilakukan dengan membuat persamaan dan operasi secara acak
- Banyak operasi kueri dan perubahan dibuat hampir sama, dengan urutan acak
- Setiap nilai acak yang dibuat memiliki distribusi uniform dengan batasan 0 sampai N untuk indeks persamaan dan 0 sampai 10006 untuk nilai k dan b
- Setiap uji coba dilakukan dengan 50 kasus uji kemudian diambil waktu rata-ratanya

PENGARUH BANYAK VERTEX TERHADAP WAKTU

- Banyak vertex bervariasi antara 3 sampai 30000 dengan rentang logaritmik berbasis 10. Banyak operasi dibuat tetap yaitu 100000.

No	Banyak Vertex	Waktu (detik)
1	3	0.072
2	30	0.111
3	300	0.145
4	3000	0.179
5	30000	0.244

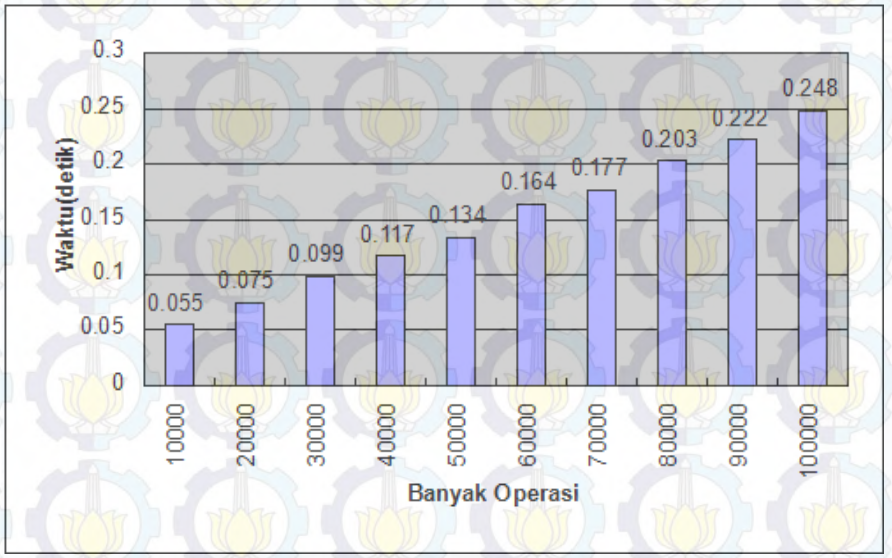


- Pertumbuhan waktu yang dibutuhkan program linear seiring dengan pertumbuhan vertex yang logaritmik. Hal tersebut membuktikan bahwa pertumbuhan waktu terhadap banyak vertex logaritmik.

PENGARUH BANYAK OPERASI TERHADAP WAKTU

- Banyak vertex dibuat tetap yaitu 30000. Banyak operasi bervariasi antara 10000 sampai 100000 dengan rentang 10000.

No	Banyak Vertex	Waktu (detik)
1	10000	0.055
2	20000	0.075
3	30000	0.099
4	40000	0.117
5	50000	0.134
6	60000	0.164
7	70000	0.177
8	80000	0.203
9	90000	0.222
10	100000	0.248



- Pertumbuhan waktu yang dibutuhkan program linear seiring dengan pertumbuhan banyak operasi. Hal tersebut membuktikan bahwa pertumbuhan waktu terhadap banyak operasi linear.

KERANGKA PRESENTASI

- Pendahuluan
- Ilustrasi
- Uji Coba dan Evaluasi
- Kesimpulan

KESIMPULAN

1. Implementasi yang telah dilakukan dapat menyelesaikan permasalahan sistem persamaan kongruen yang dinamis dengan benar
2. Struktur data link cut tree mempunyai amortized time complexity O(log n) untuk setiap operasi akses, mencari root, penghapusan edge, penyisipan edge, dan getval

PERTANYAAN?